Universidad de Alcalá
Escuela Politécnica Superior

Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

ENTORNO DE CONTROL DE SISTEMAS IIOT BASADO EN MQTT IMPLEMENTADO EN LABVIEW

Autor: Ana María Vera Martín

Tutor/es: Francisco Javier Rodríguez Sánchez

2019

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

ENTORNO DE CONTROL DE SISTEMAS IIOT BASADO EN MQTT IMPLEMENTADO EN LABVIEW

Autor: Ana María Vera Martín

Tutor: D. Francisco Javier Rodríguez Sánchez

TRIBUNAL:

Presidente: José Antonio Jiménez Calvo

Vocal 1º: Sirona Valdueza Felip

Vocal 2º: Francisco Javier Rodríguez Sánchez

FECHA: Septiembre 2019

A mis padres,

A toda mi familia,

A mi marido Carlos,

Y a mis dos amores, Alan & Fénix.

Me gustaría agradecer a toda la gente que me ha ayudado y apoyado a lo largo de todos estos años, tanto académicamente como en lo personal, en especial, a:

Mis padres, sin vuestra ayuda y comprensión a lo largo de los años y ese apoyo en todo momento no habría sido posible llegar a donde estoy,
Mi hermano, aún con nuestros altibajos siempre hay palabras de ánimo y fuerza para terminar lo que parecía interminable,
A mi familia política por el apoyo y la fuerza a lo largo de los años,
A Julio Pastor, porque gracias a sus palabras y franqueza supe poner un punto y coma en mis estudios cuando todo parecía imposible y retomarlos con más fuerza cuando fui capaz de hacerlo, muchas gracias,
A todos esos docentes que se han esforzado porque aprendiera, no solo académicamente sino personalmente, gracias por todos esos años,
A mi marido Carlos Girón, podría estar agradeciéndote la vida entera y aún me faltaría tiempo. Gracias por ser como eres y apoyarme en todo momento, en lo bueno y en lo malo, en las risas y en los llantos, hacerme ver que TODO es posible si se desea de verdad y no dejarme tirar la toalla. Sabes que esto es gracias a ti.
A mis perros, Alan & Fénix, que llegaron en los peores momentos para llenarlo todo de risas y alegría. Sin su cariño y amor no habría podido conseguir todo esto.
Y, a toda esa gente que en algún momento se ha cruzado en mi vida y me ha aportado su pequeño granito de arena.

GRACIAS

ÍNDICE

Resumen

En este Trabajo Fin de Grado se ha realizado la implementación de un entorno de control de un sistema *Industrial Internet of Things* basado en *Message Queuing Telemetry Transport* con *Labview*.

Para ello se estudia la definición de sistemas IIoT, los paradigmas de comunicación empleados y, en especial, el protocolo MQTT.

Se ha empleado la herramienta de programación *Labview* y, con la ayuda de la librería MQTT encontrada, se desarrollan simuladores de dispositivos compatibles con el protocolo y se realiza el control del sistema.

Por último, se implementa el sistema completo, dispositivos y control, con un *Broker* MQTT.

Summary

In this Final Degree Project, the implementation of a control environment of an Industrial Internet of Things system based on Message Queuing Telemetry Transport with Labview has been carried out.

For this, the definition of IioT systems, the communication paradigms used and, especially the MQTT protocol are studied.

The Labview Programming tool has been used and, with the help of the MQTT library found, device simulators compatible with the protocol are developed and system control is performed.

Finally, the complete system, devices and control, is implemented with an MQTT *Broker*.

Palabras clave

Internet of Things, Industrial Internet of Things, Labview, Message Queuing Telemetry Transport.

Glosario de acrónimos y abreviaturas

Acrónimo	Español	Ingles
TFG	Trabajo Fin de Grado	
Labview	Laboratorio Virtual de Bancos de Trabajo para Ingenieria de Instrumentación	Laboratory Virtual Instrument Engineering Workbench
MQTT	Transporte de Telemetria de colas de mensajes	Message Queuing Telemetry Transport
IoT	Internet de las Cosas	Internet of Things
IIoT	Internet Industrial de as Cosas	Industrial Internet of Things
NI	National Instrument	National Instrument
M2M	Maquina a Maquina	Machine to Machine
UIT-T	Union internacional de Telecomunicaciones, Sector de Estandarización	International Telecommunication Union, Telecommunication Standardization Sector
TCP	Protocolo de Control de Transmision	Transmission Control Protocol
IP	Protocolo de Internet	Internet Protocol
QoS	Calidad de Servicio	Quality of Service
IEEE	Instituto de Ingenieros Eléctricos y Electrónicos	Institute of Electrical and Electronics Engineers
ETSI	Intituto dEuropeo de Normas de Telecomunicaciones	European Telecommunication Standards Institute
UIT	Union internacional de Telecomunicaciones	International Telecommunication Union

CASAGRAS	Coordinacion y acción de apoyo para actividades y estandarización global rlacionadas con RFID	Coordination and Support Action for Global RFID-related Activities and Standardization
HTTP	Protocolo de Transferencia de Hipertexto	Hipertext Tranfer Protocol
REST	Transferenia de estado representacional	Representational State Transfer
CoAP	Protocolo de aplicación restringido	Constrained State Protocol
MXPP	Mensajeria extensible y protocolo de presencia	Extensible Messaging and Presence Protocol
DDS	Servicio de distribución de datos	Data Distribution Service
Stromp	Protocolo de mensajeria orientado a texto simple o continuo	Simple or Streaming Text Oriented Messaging Protocol
AMQP	Protocolo de colas de mensaje avazado	Advanced Message Queuing Protocol
TLS	Capa de seguridad de transporte	Transport Layer Security
UDP	Protocolo de datagramas de usuario	User Datagram Protocol
SSL		Secure Sockets Layer
GPS	Sistema de posicion global	Global Positioning Global
RFID	Identificador de Radiofrecuencia	Radio Frequency Identification
QR		Quick Response
FPGA	Array de puertas programables	Field-programmable gate array

VI	Instrumento Virtual	Virtual Instrument
PF	Panel Frontal	
DB	Diagrama de Bloques	
ISO	Organización internacional de estandarizacion	International Organization for Standardization
PLC	Controlador Logico Programable	Programable Logic Controller
MAX	Explorador de medidas y automatización	Measurement & Automation Explorer
VISA	Arquitectura de software de instrumentos virtuales	Virtual Instrument Software Architecture
ASCII		American Standard Code for Information Interchange
TDMS	Test de flujo de intercambio de datos	Test Data Exchange Stream
I/O	Entrada /Salida	In/Out
DUP	Indicador de duplicado	Duplicate message
SSOO	Sistemas Operativos	

Memoria

1 Introducción

El objetivo principal de este Trabajo Fin de Grado, TFG, es familiarizarse con los entornos de control de sistemas IIoT, empleando el protocolo de comunicación MQTT e implementarlo con la herramienta de programación de National Instrument (NI) *Labview*.

Primero se realiza el estudio del estado del arte, donde se muestran las diferentes definiciones de IoT e IIoT y las principales diferencias entre ambos. Se introduce un pequeño resumen de los dos paradigmas de comunicaciones más importantes y sus diferencias, cliente/servidor y publicador/suscriptor, y se mencionan los diferentes protocolos más empleados en IoT e IIoT, para introducir la importancia que tiene el protocolo MQTT y por qué es tan interesante para la Industria y los dispositivos IIoT.

En sucesivos capítulos, se realiza un estudio de la herramienta de programación de NI, *Labview*, donde se estudian los conceptos básicos y diferentes estrategias de programación y depuración, y se entra en profundidad en el protocolo de comunicación MQTT, comprendiendo su intercambio de mensajes y las características del mismo.

Por último, se comentan las librerías del protocolo asociadas a *Labview* encontradas, comprendiendo el procesamiento y envío de cada mensaje. Este punto es muy importante dado que será la base para la realización de un caso de uso básico de un entorno de control de un sistema IIoT.

Este caso de uso, se basa en diseñar y realizar una aplicación en *Labview* que implemente el protocolo MQTT y aplicarlo en un entorno de control de sistemas IIoT, simulando el comportamiento de sensores y actuadores compatibles con el protocolo.

A grandes rasgos, los pasos empleados son:

- Definición de las especificaciones del sistema.
- Modificaciones en las librerías MQTT para adecuarlas al funcionamiento deseado.
- Diseño de los simuladores de dispositivos, que se emplearían en el sistema.
- Realización de la aplicación de control del sistema en *Labview*.
- Despliegue de un *Broker* MQTT, para realizar la interconexión de los dispositivos y el control del sistema.
- Implementación y pruebas de la simulación del sistema completo.

2 Estado del arte

2.1 Introducción

A día de hoy, es difícil encontrar algún objeto utilizado en la vida cotidiana y en el que no aparezcan las palabras Internet of Things (o en castellano Internet de las Cosas) relacionadas con él.

Como explican en [1], Internet está evolucionando y cambiando continuamente: en 1969 se estableció la primera conexión de red entre UCLA y Stanford, y durante la mayor parte de su vida, su principal función ha sido realizar la comunicación entre humanos.

En 1990, John Romkey conectó a Internet la primera tostadora, haciendo que este se encendiera y apagara en remoto [2].

En 1999, Kevin Ashton acuñó el término "Internet of Things" en una presentación en Procter & Gamble, y a partir de esa fecha, aparece una filosofía de comunicación por la que se quiere dotar a los objetos de ciertas capacidades de comunicación, con los humanos y con otros objetos. Esto ha dado paso a que Internet realice un aprendizaje M2M.

En la actualidad, la mayoría de aparatos electrónicos y/o aplicaciones software que utilizamos habitualmente, están interconectados entre sí, haciendo posible que se pueda acceder a los datos recopilados de cada uno de ellos, desde cualquier ordenador o dispositivo portátil.

La idea fundamental de IoT, es la interconexión del mundo físico, el mundo virtual y el mundo digital, de manera que la sociedad pueda beneficiarse de esa interconexión, tal y como se muestra en la Figura 1.

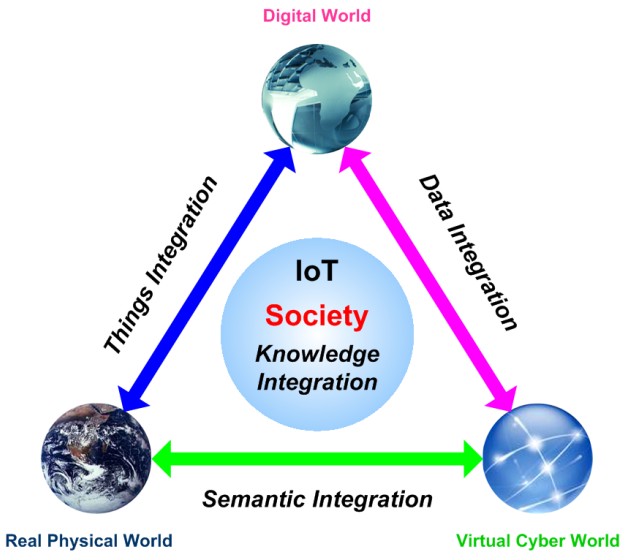


Figura 1: Filosofía IoT [3]

2.2 Qué es IoT

La recomendación de la UIT-T, Y.2060 [4], define IoT como “una infraestructura global de la sociedad de la información, que propicia la prestación de servicios avanzados mediante la interconexión de objetos (físicos y virtuales) gracias a la interoperabilidad de tecnologías de la información y la comunicación presentes y futuras”

Se puede definir a grandes rasgos IoT, como la infraestructura encargada de interconectar el mayor número de objetos, tanto físicos como virtuales, entre sí y con el ser humano. Esta interconexión se puede establecer, bien a través de pequeñas redes privadas o a través de Internet. De este modo, se puede monitorizar el estado de cada uno de estos objetos desde cualquier lugar, en cualquier momento.

Para ello, se aprovecha la capacidad de identificación, adquisición de datos, procesamiento y comunicación, que poseen los dispositivos electrónicos que están en los objetos y se emplean otros dispositivos provistos de mayor inteligencia para poder ofrecer servicios a todos los tipos de aplicaciones IoT, garantizando la privacidad y la seguridad de los datos adquiridos.

Las redes de comunicaciones son las encargadas de transferir los datos adquiridos por los dispositivos a las aplicaciones y a otros dispositivos. Estas redes deben ofrecer una transferencia de datos fiable y eficiente entre dispositivos y aplicaciones.

Las características fundamentales de IoT tal y como se describen en [4], son:

- Interconectividad: esta es la característica que dota a IoT de toda su importancia, es la característica que debe permitir la compatibilidad y el acceso a la infraestructura mundial de la información y la comunicación.
- Servicios relacionados con objetos: IoT debe proporcionar servicios relacionados con objetos dentro de las restricciones de esos objetos, como la privacidad y seguridad de los datos obtenidos por los mismo.
- Heterogeneidad: dispositivos empleados en IoT están basados en diferentes plataformas hardware y redes, por lo tanto, podrán interactuar entre sí y con otras plataformas de servicios a través de redes diferentes.
- Cambios dinámicos: tanto el estado de los dispositivos (reposo, activo, conectado, etc.), como el contexto (ubicación, velocidad, ...) o el número de dispositivos pueden variar de forma dinámica.
- Escalabilidad: el número de dispositivos IoT interconectados puede incrementarse incluso un orden de magnitud durante los próximos años, lo que requerirá mayor comunicación entre estos dispositivos y hará necesaria la gestión, interpretación y manipulación de los datos generados de manera eficiente y segura.

Por otro lado, todo dispositivo IoT debe cumplir un requisito esencial que es que disponga de capacidades de comunicación. Además de existir otros requisitos de alto nivel como se describe en [4]:

1. Conectividad basada en la identificación: Todo dispositivo que quiera ser utilizado en la infraestructura IoT, debe estar identificado, de manera que, para que sea heterogéneo habrá que procesar identificadores de manera unificada.
2. Compatibilidad: se debe garantizar la compatibilidad entre sistemas heterogéneos, para el uso de diferentes servicios y datos.
3. Capacidades basadas en la ubicación: las comunicaciones y servicios relacionados con los objetos pueden depender de la ubicación de los mismo y/o de los usuarios, por lo tanto, deben cumplir la seguridad y las leyes que rijan cada lugar.
4. Seguridad: todos los objetos interconectados presentan amenazas de seguridad. De ahí deriva la importancia de mantener requisitos de seguridad para mantener la integridad, la confidencialidad y la autenticidad de los datos.
5. Protección de la privacidad: algunos objetos obtienen información privada de los usuarios, por lo que es importante dar soporte a la protección de la privacidad durante toda la manipulación de los datos obtenidos, desde la transmisión, al procesamiento o almacenaje de estos.
6. Autoconfiguración (*plug and play*) de dispositivos y servicios: Los dispositivos deben soportar su configuración automática dentro de la infraestructura IoT. Los servicios se deben poder configurar a partir de datos obtenidos de los objetos por los clientes o los operadores de servicios.
7. Capacidad de administración: IoT debe ser capaz de trabajar de forma autónoma y sin necesidad de la intervención humana, pero el proceso global de funcionamiento, deben poder gestionarlo las partes competentes a tal fin.

La arquitectura de Internet tiene cinco capas (TCP/IP), pero con el incremento de objetos que se están conectando a internet, se hace patente la necesidad de emplear otro tipo de arquitectura para poder cumplir con las características fundamentales de escalabilidad, modularidad, interoperabilidad, QoS, etc.

En [5] aparecen varias propuestas de arquitectura dependiendo de la organización que trata de estandarizar IoT:

- IEEE, en 2014, propone una arquitectura en tres capas, tal y como se muestra en la Figura 2. Propone una capa de sensado, otra capa de redes y comunicaciones de datos y una última capa de aplicación.

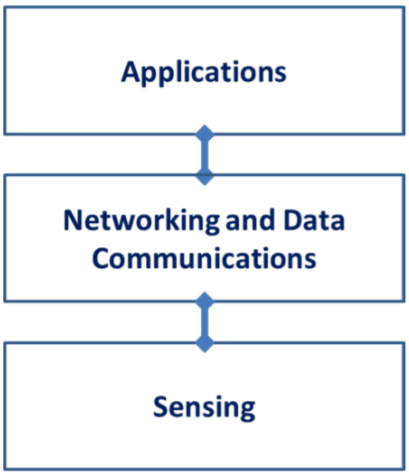


Figura 2: Arquitectura IoT de tres capas (IEEE)

- ETSI, propone un modelo de arquitectura basado en comunicación M2M. Tal y como se muestra en la Figura 3, hay una parte de la arquitectura que se encuentra en el dominio físico y otra en el dominio virtual. En [6], se explica con mayor detalle cada una de las capas.

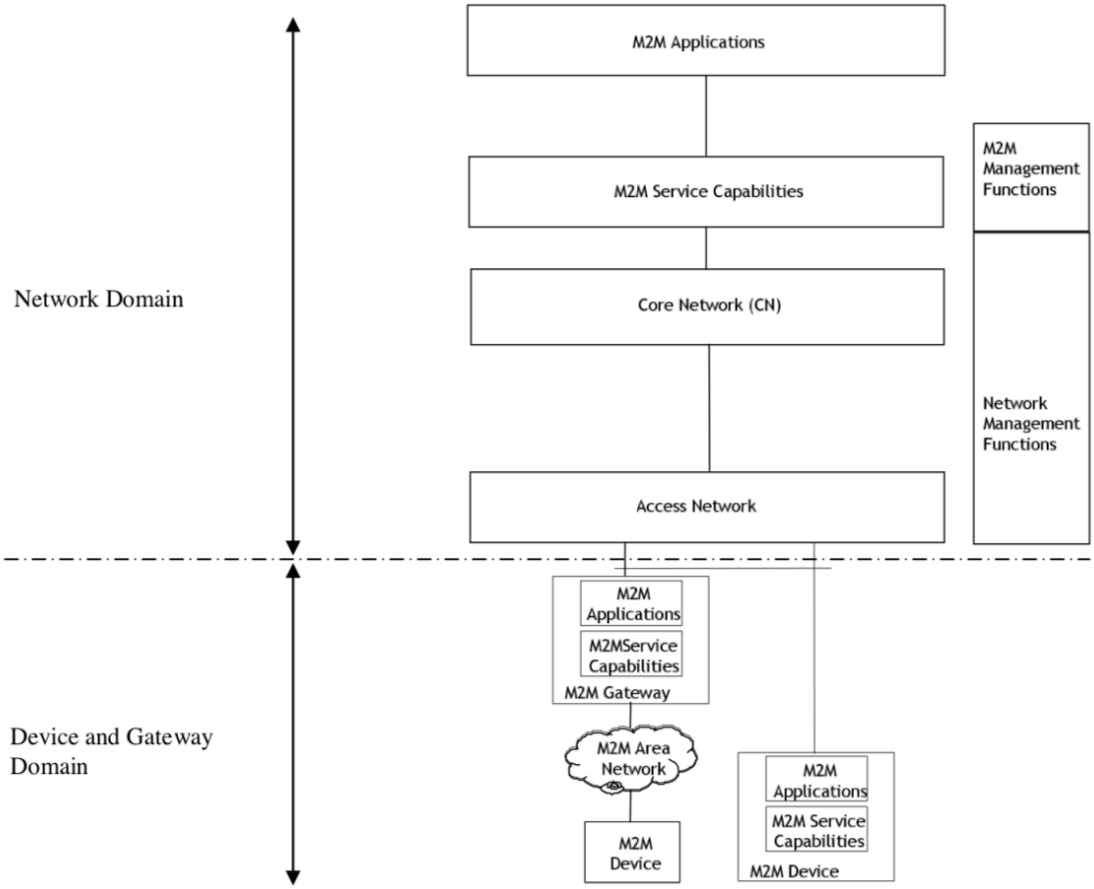


Figura 3: Arquitectura M2M (ETSI)

- oneM2M, propone una arquitectura funcional con una serie de requisitos [7]. En la Figura 4, se puede diferenciar entre el usuario final, la red de operadores, los servicios de conectividad y la solución M2M. A su vez, también describe su modelo de tres capas, tal y como se muestra en la Figura 5, siendo estas, la capa de aplicación, la capa de servicios comunes y la capa de servicios de red.

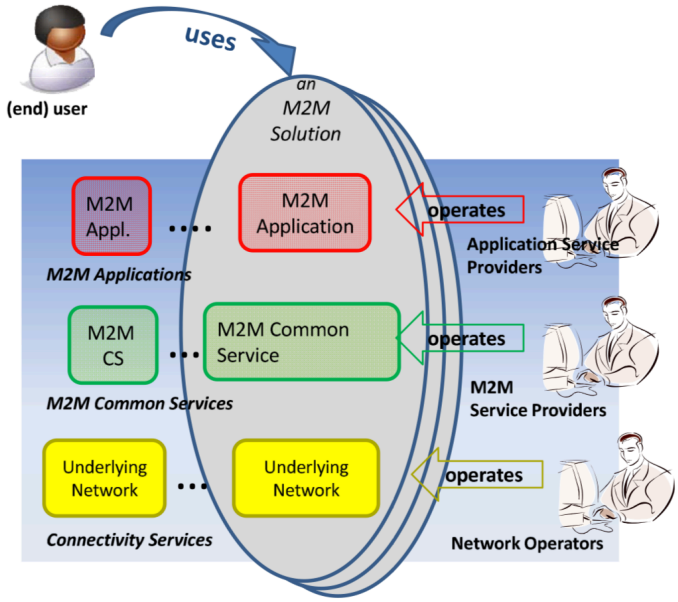


Figura 4: Arquitectura funcional (oneM2M)

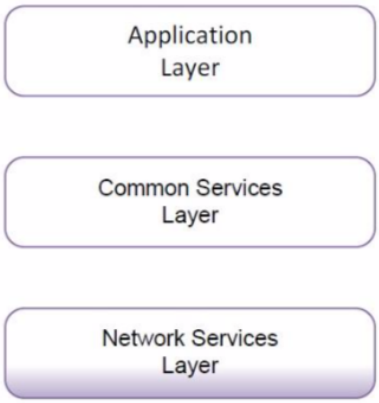


Figura 5: Modelo de capas (oneM2M)

- UIT, en 2016, realiza una recomendación [4], donde propone que el modelo de referencia sea el que se muestra en la Figura 6. En él observamos que establece cuatro capas y dos capacidades que están relacionadas con estas cuatro capas: capa de aplicación, capa de apoyo a servicios y aplicaciones, capa de red y capa de dispositivo. Las capacidades asociadas a las capas son: Capacidad de seguridad y capacidad de gestión.

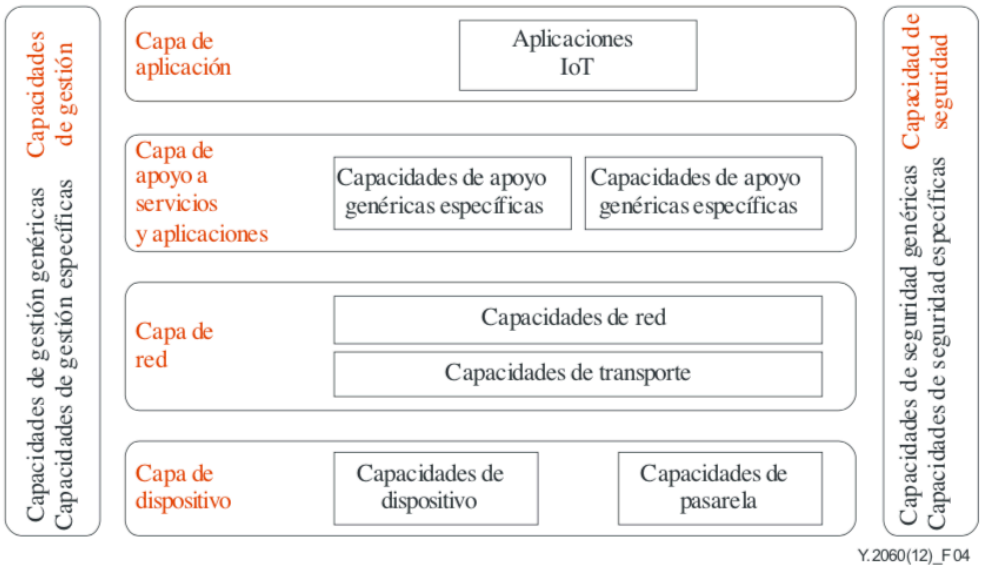


Figura 6: Modelo de referencia IoT (UIT)

- El grupo CASAGRAS en 2009 [8], propone que el modelo de arquitectura de IoT sea el mostrado en la Figura 7, donde establece tres capas: Capa física, capa de enlace entre objetos y los gestores de datos, y la capa de gestores de datos.

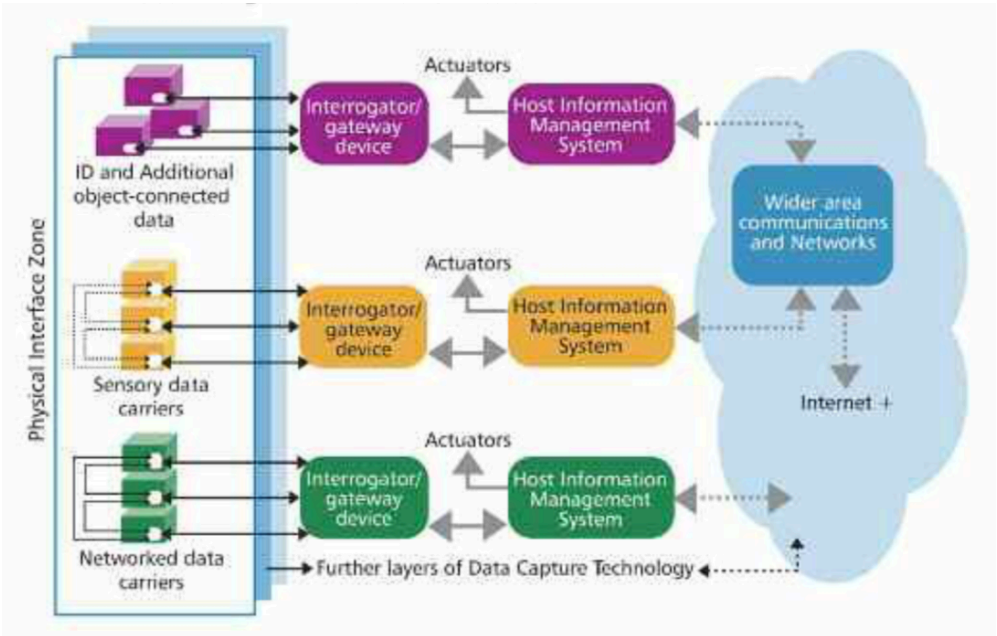


Figura 7: Modelo de arquitectura IoT (CASAGRAS)

A día de hoy, todas estas arquitecturas se han quedado en propuestas. Dependiendo del proyecto, los dispositivos, el nivel sociocultural y/o la zona geográfica donde se quiera conectar un dispositivo a Internet, podremos hablar de una arquitectura o de otra.

2.3 Qué es IIoT

El desarrollo de la tecnología y uso de la automatización industrial y la inteligencia integrada desde mediados del siglo XX, ha propiciado la aparición de objetos IoT.

El uso de estos objetos IoT en el ámbito industrial, da como resultado un nuevo concepto de negocio llamado Industria 4.0 que se basa en objetos IIoT. Industria 4.0 es capaz de mejorar los controles de calidad, es respetuoso con el medio ambiente y puede realizar un seguimiento de la cadena de suministro, entre otras cosas, por lo que hace que el potencial de este concepto de negocio sea muy grande.

A grandes rasgos, IIoT es la aplicación de instrumentación y sensores y otros dispositivos a maquinaria y vehículos en los sectores de transporte, energía e industria, y atención médica. [9]

El objetivo de IIoT según [10], es optimizar la producción industrial a través de la conexión entre dispositivos y un centro de datos para poder saber lo que sucede en tiempo real, pudiendo controlar la producción u obtener datos que nos ayuden a optimizar el negocio. Para ello se incorpora una red de máquinas y dispositivos con mayor inteligencia que tal y como se asegura en [11], deben poderse configurar y manejar a través de un software.

Se puede decir que IIoT está segmentada en tres bloques, tal y como indican en [11]. En la Figura 8, se puede distinguir una zona inteligente, compuesta por sensores y actuadores, el sistema de sistemas, y la conectividad y análisis de datos.

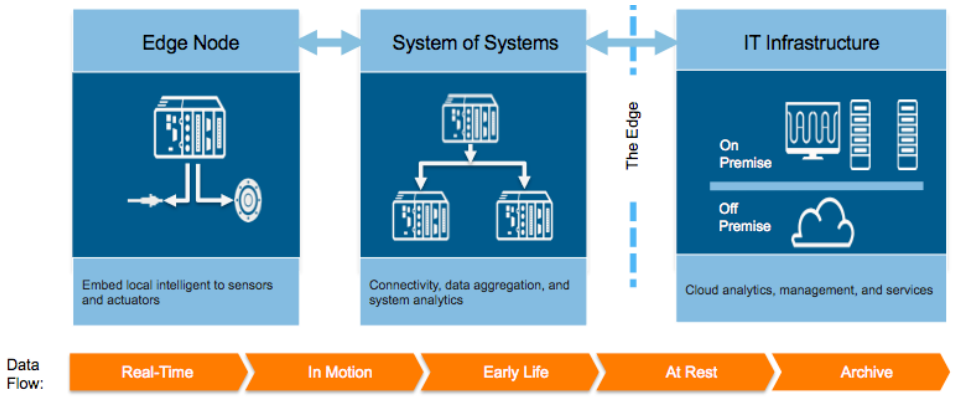


Figura 8: Segmentación IIoT según [11]

- En la zona inteligente, se encuentran los sensores y actuadores. Son similares a los empleados en IoT, aunque en algunos casos, deben ser más robustos, y con rangos de trabajo más amplios.
- En la zona de sistema de sistemas, se engloban los protocolos de comunicación, la extracción de datos y el método de análisis de los mismos.

- En la zona de infraestructura IT, se encuentra los análisis de los datos, la gestión de los mismos y los servicios que utilizaran los datos.

Muchos expertos aseguran que la introducción de los objetos IIoT en la industria, supondrán una revolución en el mundo industrial llamada *Industria 4.0.* Esta integración de la tecnología de la información es el soporte para la optimización e interacción de los procesos de investigación y desarrollo, diseño, producción, logística y prestación de servicios asociados a la empresa, tal y como afirman en [12].

Industria 4.0 engloba varios campos dentro de la industria, tales como: edificios, productos y logística inteligentes, Big Data, Smart Grids, Smart Mobility, Cloud Computing, automatización robótica, realidad aumentada, etc., tal y como se muestra en la Figura 9.

Figura 9: Ejemplo de campos abarcados en Industria 4.0 [12]

Como sugieren en [13], las principales características de IIoT en el marco de la Industria 4.0 son:

1. La inteligencia, los elementos IIoT de la fábrica, en muchos casos deben ser capaces de poder interoperar entre ellos y actuar de manera independiente en función de una serie de parámetros preestablecidos.
2. En cuanto a la arquitectura, la tendencia que está surgiendo, es que sea una "arquitectura basada en eventos" y que sea capaz de tomar decisiones en función de los datos y pueda aprender de ellos.
3. Sistema complejo: al crecer el número de objetos a interconectar, se hace complejo el sistema al existir gran cantidad de pequeñas subredes dentro de la gran red que es Internet.

Uno de los problemas que se ha encontrado la Industria en su avance hacia la llamada Industria 4.0 es la diversidad de protocolos de comunicación para enviar y recibir datos de los diferentes objetos físicos y virtuales. A día de hoy, hay

muchos protocolos que están en uso, pero hay uno de ellos que está despuntando entre los demás, MQTT.

Otro de los grandes desafíos es la seguridad que rodea a la implementación de IIoT. Los datos obtenidos, para muchas empresas, son activos críticos y hay que *securizarlos*, dado que el acceso remoto, el almacenamiento y la transmisión de los mismos son puntos vulnerables para dichas empresas.

La seguridad en dispositivos IIoT está íntimamente relacionada con el tipo de conectividad y el protocolo de comunicación. Según [14], se pueden evitar algunas vulnerabilidades haciendo que los dispositivos IIoT estén conectados a la red únicamente cuando vayan a ser utilizados. Pero en algunos casos, no es posible desconectar los dispositivos de la red.

Un método para *securizar* el acceso a los dispositivos es realizando la seguridad a través de aplicaciones o a través de un dispositivo puente conectado al anterior, de modo que se complique el acceso al dispositivo principal.

2.4 Diferencias entre IoT e IIoT

La principal diferencia con IoT, es que IIoT se centra en conectar máquinas y dispositivos en industrias, transportes, servicios de energía y cuidado de la salud, mientras que IoT se centra en dispositivos y otras aplicaciones que no crean situaciones de emergencia si algo sale mal, como monitorización de objetos de la vivienda. En la Figura 10, se puede observar las competencias de IoT e IIoT.

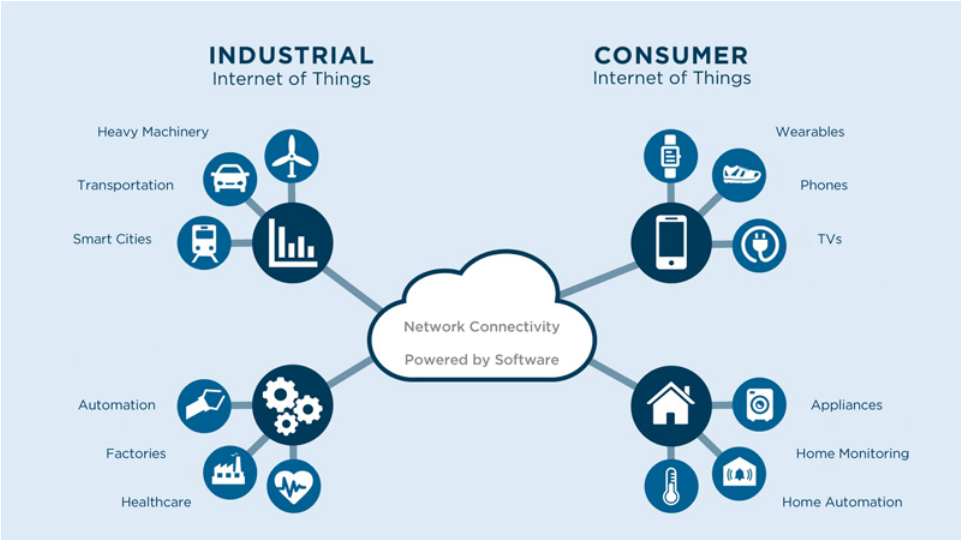


Figura 10: IIoT vs IoT

Tanto los dispositivos IoT como los dispositivos IIoT, generan variables que pueden ser monitorizadas desde cualquier lugar, ya sea su propia red o una red pública. Hay dispositivos que generan una gran cantidad de variables y deben analizar y procesar los datos antes de enviarlos, para no saturar al sistema central.

En el caso de los dispositivos IIoT, son más sofisticados para permitir controles automatizados más sensibles, por lo que el rango de trabajo será más preciso y estable. A su vez, suelen ser más robustos y, por lo tanto, suelen ser más costosos, dado que habitualmente estarán sometidos a unas condiciones diferentes que los dispositivos IoT.

Por el contrario, los dispositivos IoT al tener menor precio tendrán una mayor evolución tecnológica.

Algunas de las aplicaciones de IIoT necesitan trabajar con cifrado de datos y protegerlos ante posibles intrusiones y ataques, mientas otras menos críticas para la seguridad humana, no necesitan trabajar con dicha seguridad.

Como conclusión, la diferencia entre IoT e IIoT se encuentra en el impacto de riesgo humano. IoT tiene un bajo impacto, mientras IIoT puede provocar una situación de emergencia y poner en peligro la vida del ser humano. Por lo que, para IIoT se necesita más fiabilidad, más seguridad y más coste.

2.5 Paradigmas de comunicación

Para interconectar los dispositivos y máquinas que trabajan englobadas en la filosofía IoT e IIoT, existen diversos protocolos. Los podemos agrupar en dos tipos de paradigma de comunicación: cliente/servidor y publicación/suscripción.

2.5.1 Cliente/servidor:

Es un modelo de comunicación en el que se vinculan varios dispositivos electrónicos a través de una red. En la Figura 11 se muestra un posible ejemplo de conexionado de un servidor con tres clientes diferentes, con los cuales puede trabajar de manera aislada o simultáneamente.

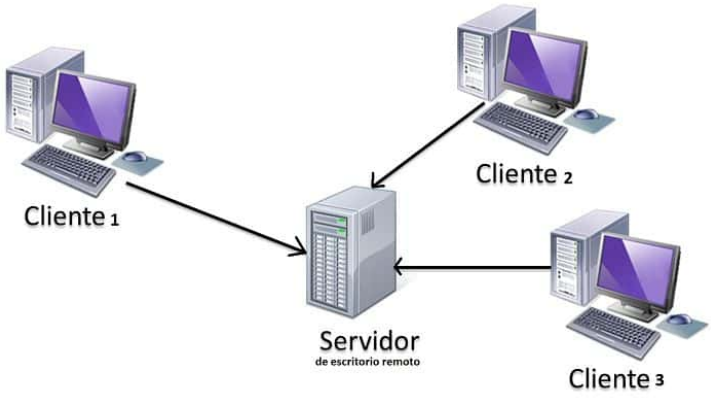


Figura 11: Ejemplo de conexionado cliente/servidor [15]

Las tareas se distribuyen entre los proveedores de servicios (servidores) para poder satisfacer las peticiones de los clientes. Por lo tanto, se puede repartir la capacidad de procesamiento entre varios dispositivos, actuando estos como

servidores. Cada servidor puede procesar peticiones de uno o varios clientes simultáneamente.

Un ejemplo de comunicación entre cliente y servidor se muestra en la Figura 12, donde se puede apreciar la comunicación entre un cliente y un servidor durante una sesión. En la primera petición, el cliente establece la comunicación y la primera respuesta del servidor confirma el establecimiento de la comunicación. El número de peticiones y respuestas dependerá del servicio solicitado.

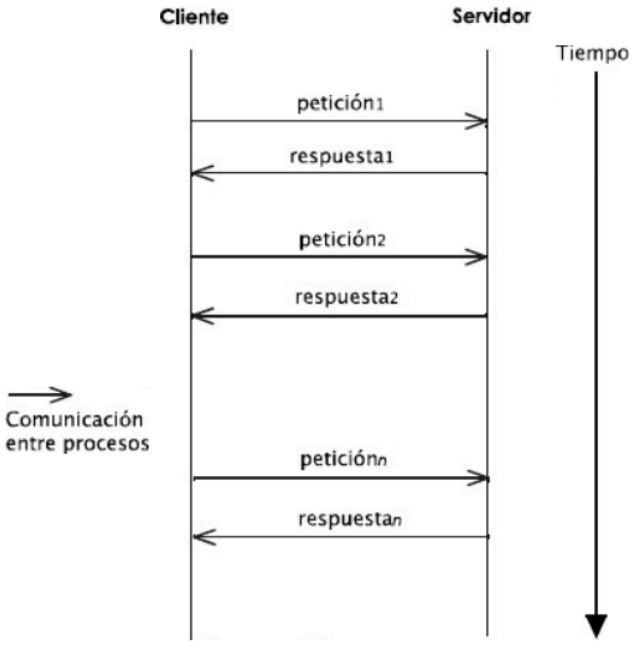


Figura 12: Patrón comunicación cliente/servidor en una sesión [15]

Un ejemplo de servicio de este modelo de comunicación es “el correo electrónico”, donde el servidor web se encuentra en un dispositivo y provee el servicio a los dispositivos que lo requieren. Otro ejemplo de servicio es “Daytime”, donde el servidor responde al cliente que lo solicita con una marca de tiempo.

Existen unos protocolos asociados a cada servicio. Algunos servicios tienen varios protocolos asociados, especifican cómo se produce el dialogo en una sesión, la sintaxis y la semántica de la comunicación y la acción esperada tanto por el cliente como por el servidor.

El cliente, por lo tanto, debe conocer de antemano quién es el servidor y sus datos para poder solicitar la información.

Hay que tener en cuenta que, si una red tiene gran cantidad de clientes y pocos servidores, si los clientes realizaran una solicitud simultáneamente, pueden causar dificultades de procesamiento para el servidor.

2.5.2 *Publicación/suscripción:*

Es un modelo de comunicación en el que los dispositivos publican datos de un evento en un intermediario, al que se suscriben los consumidores. No es necesario que tanto publicadores como suscriptores se encuentren en la misma red, ni tan siquiera que conozcan la ubicación los unos de los otros.

Las partes implicadas en la comunicación son:

- Un consumidor o suscriptor, recibe información de los *Topics* a los que está suscrito.
- Un publicador, puede publicar tantos *Topics* diferentes como crea preciso.
- Un *Broker* o mediador, es el encargado de recoger la información de los *Topics* y enviarla a los suscriptores.

El suscriptor puede estar suscrito a uno o varios Topics, de un mismo publicador o de varios. El publicador es el responsable de definir los diferentes tipos de Topics a los cuáles se pueden suscribir los suscriptores. El suscriptor y el publicador pueden coexistir en un mismo dispositivo. En la Figura 13 se muestra un ejemplo de conexión publicador/suscripción, con las partes involucradas en la comunicación.

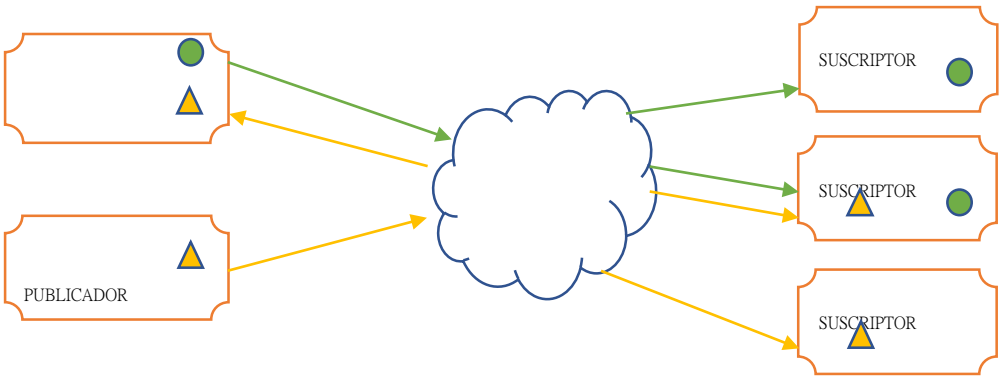


Figura 13: Ejemplo de conexión publicador/suscriptor

La comunicación entre estas tres partes implicadas se realiza a través de canales. Se especifican dos tipos de canales diferentes dentro de la comunicación, vistos desde el mediador o *Broker*:

- Canal de entrada, se trata de un sistema unidireccional que establece la comunicación entre el publicador o editor y el *Broker*. Se crea un canal de entrada diferente por cada *Topic* publicado, por cada uno de los publicadores.
- Canal de salida, se trata de un sistema unidireccional que establece la comunicación entre el *Broker* y cada uno de los suscriptores. Se crean tantos canales como suscriptores están suscritos al *Topic* en cuestión.

En la Figura 14, se observa la comunicación de dos *Topics*, uno rojo y otro azul. Cada uno de los Editores manda su *Topic* al *Broker* o sistema de mensajes a

través de un canal de entrada propio y posteriormente el *Broker* envía cada uno de los *Topics* a los suscriptores suscritos a cada uno de ellos a través de un canal de salida propio.

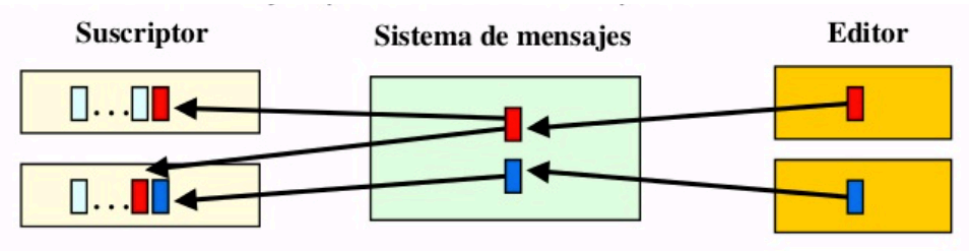


Figura 14: Modelo de mensajes publicación/suscripción

Este paradigma de comunicación es la mejor opción cuando se desconoce la infraestructura del sistema, dado que no hay interacción entre publicadores y suscriptores y viceversa.

Además, proporciona menos ocupación del ancho de banda, puesto que sólo se transmite información a los que realmente les interesa recibirla.

El protocolo que utilice este paradigma será el que tome decisiones como: el momento de lectura de la información, el tiempo que el canal permanece abierto, el tipo de información que transmite, el formato de la misma, etc.

2.6 Protocolos de comunicación

Existen diferentes protocolos que se encuentran englobados en cada una de estas categorías: [16] [17]

- HTTP: Es un protocolo cliente/servidor. Se trata de código abierto y por lo tanto es muy utilizado. Es muy útil para enviar grandes cantidades de datos a la vez. No es útil para aplicaciones en tiempo real. La seguridad se realiza a través de SSL/TLS.
- REST: Se utiliza para intercambiar datos entre aplicaciones y para integrar aplicaciones que pertenecen a diferentes dominios. Se trata de un protocolo cliente/ servidor. Cada solicitud de cliente debe mandar toda la información necesaria para comprender la solicitud. No se almacenan datos. [18]
- CoAP: Es un protocolo M2M, pensado para acercar HTTP a dispositivos y redes restringidas. Se trata de un protocolo con arquitectura cliente/servidor. La comunicación se realiza a través de UDP. [19]
- MXPP: Emplea mensajes en formato XML. Permite a los usuarios enviar mensajes en tiempo real, y gestiona la presencia del usuario. Permite enviar y recibir mensajes de máquinas y/o dispositivos. [20]
- DDS: Protocolo de tipo publicación/suscripción concebido para sistemas de tiempo real. La comunicación entre los nodos se realiza a través de UDP. [21]

- Stomp: Protocolo orientado a mensajería de texto. Es un protocolo cliente/servidor que funciona sobre TCP. [22]
- AMQP: Es un protocolo M2M para aplicaciones distribuidas de tipo publicación/suscripción. Proporciona características como el enrutamiento y gestión de colas dentro del protocolo. Y ofrece seguridad a través de la autenticación y cifrado mediante TLS. La comunicación se realiza a través de TCP/IP. [23]
- MQTT: Es un protocolo de tipo publicación/suscripción de nivel de aplicación. Es recomendable usarlo con TLS para asegurar las comunicaciones. La comunicación se realiza a través de TCP/IP.

Es importante elegir el protocolo que mejor se adapta a las necesidades y seleccionar la tecnología más apropiada para cada caso de uso.

Este TFG ce centra en el protocolo de comunicación MQTT, por su paradigma de comunicaciones, publicador/suscriptor, y porque es uno de los más empleados en la industria debido a la baja sobrecarga del canal que aporta el protocolo, lo que hace que sea muy interesante su uso en dispositivos que tienen poca capacidad computacional.

2.7 Dispositivos y objetos IoT

La Recomendación UIT-T Y.2060 [4] define el dispositivo como “una pieza de equipo con las capacidades obligatorias de comunicación y las capacidades opcionales de detección, de accionamiento y de adquisición, almacenamiento y procesamiento de datos”.

El requisito mínimo que ha de tener un dispositivo para ser IoT es que disponga de capacidad de comunicarse con otros dispositivos y/o aplicaciones.

En [4], se establece una clasificación de dispositivos IoT, de modo que nos encontramos con:

- dispositivos de transporte de datos: encargados de unir, la red de transporte de datos con un objeto físico.
- dispositivos de adquisición de datos: son dispositivos de lectura / escritura que tienen capacidad para interactuar con objetos físicos, a través de dispositivos de transporte de datos, o con el dispositivo de transporte.
- dispositivos de detección y accionamiento: detectan o miden información del entorno y la convierten en señales digitales. Por lo general, estos dispositivos forman redes locales que se comunican entre sí y utilizan pasarelas para conectarse a las redes de comunicación.
- dispositivos generales: cuentan con capacidades de procesamiento y comunicación y puede comunicarse con las redes de comunicación. Incluyen equipos y aplicaciones para diferentes dominios de aplicación IoT.

En Figura 15, se muestran los diferentes tipos de dispositivos y su relación entre estos y los objetos físicos según [4].

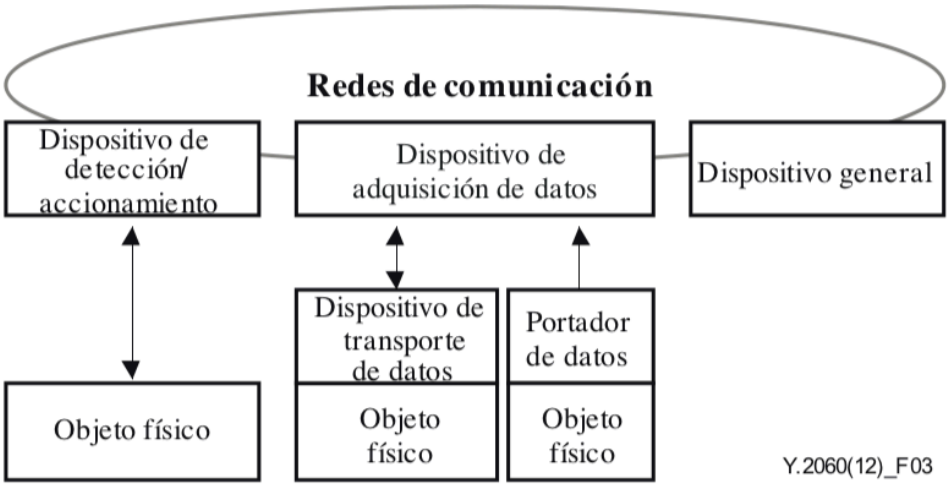


Figura 15: Tipos de dispositivos IoT

Los dispositivos IoT cada vez son de menor tamaño, lo que facilita la integración con los objetos físicos haciéndolos de menor tamaño y más cómodos para el consumo o para su uso en la industria.

Una clasificación de objetos IoT según su ámbito de aplicación:

- Vestibles: Relojes, gafas, bandas fitness y de salud, anillos, pulseras, ropa, cinturones, etc.
- Domóticos: Alarmas, cerraduras, cámaras, lavadoras, frigoríficos, televisores, control de luces, control de temperatura, control de persianas, control de riegos, etc.
- Industriales: Sensores de monitorización y control de producción, monitorización de inventario, ubicación de empleados, cámaras, control de acceso, etc.
- Automotriz: GPS, sensores para ahorro de combustible, seguro de puertas, encendido automático de luces, control de carril, estacionamiento automático, frenada de emergencia, etc.
- Ciudades inteligentes: Detectores de velocidad para monitorizar tráfico, sensores de luminaria, sensores de estructuras de edificios, cámaras de vigilancia, estacionamientos inteligentes, etc.

Según [24], los objetos y dispositivos IIoT más comunes empleados en la Industria 4.0, se clasifican en:

1) Sensores y actuadores

   Los sensores son medidores de una magnitud física o química como: temperatura, posición, luz, gas, etc.

Los actuadores son capaces de transformar energía en una acción, de manera que se pueda automatizar un proceso. Entre los más comunes están, los motores, los relés, las bombillas, etc.

Cuando estos elementos están dotados de inteligencia, se les conoce como Smart Sensors, son capaces de realizar su tarea principal y conectarse a Internet para poder enviar o recibir la información necesaria.

2) Etiquetas inteligentes

Son empleadas para obtener la trazabilidad del proceso productivo en una empresa.

Cuando un objeto llega a la cadena de producción, se le asigna una etiqueta que le identificara a lo largo de su paso por la empresa. Estas etiquetas pueden ser desde un código de barras a una etiqueta RFID. Según vaya pasando por cada una de las partes de la cadena de producción, los sensores leerán la etiqueta y le mandarán la información a un ordenador conectado a internet que la almacenará para su posterior tratamiento y realización de informes.

3) Sistemas de control embebidos

Se trata de un sistema diseñado para realizar funciones dedicadas en tiempo real, por ejemplo: leer las etiquetas inteligentes (lector RFID o QR), pulsador de emergencia con notificación de alarma, etc.

Estos dispositivos no suelen enviar todos los datos que procesan para no sobrecargar la red con información innecesaria, sino que envían el resultado de un proceso.

Entran dentro de la clasificación de IIoT porque se les conecta un módulo de comunicación para que se puedan conectar a Internet.

4) Wearables

Hacen referencia a los objetos que podemos llevar en nuestro cuerpo y que son capaces de obtener información de manera continuada del usuario.

Dentro de este conjunto se encuentran, los relojes inteligentes, las zapatillas de deporte con GPS incorporado, pulseras de fitness.

En el ámbito de la industria existen botas, gafas, guantes, cascos. Estos objetos se emplean en la prevención de accidentes laborales, controles de actividad, notificaciones, etc.

En la Figura 16, se puede observar un ejemplo de este tipo de objetos que pueden ayudar en muchos campos dentro de IIoT, entre ellos la medicina.

Figura 16: Objetos wearables

Los objetos físicos que tienen integrados dispositivos IoT, se enfrentan a un problema con la seguridad, que les hace vulnerables a ataques, según [25], algunas de estas vulnerabilidades son:

- Recursos limitados y bajo costo, las capacidades de los dispositivos pueden ser escasos y esto lleva a no poder aplicar controles avanzados de seguridad, al pretender abaratar los costos.
- Falta de control en el tratamiento de los datos, el usuario no es consciente del tratamiento que se da a los datos obtenidos en los dispositivos, y estos datos pueden pasar a terceros.
- Poca claridad en las responsabilidades, no se establece si la seguridad la debe de proporcionar el fabricante o el usuario, y esto provoca brechas en un ciberataque.

2.8 Seguridad IoT

Según la empresa americana Gartner, se espera que para 2025, haya conectados a internet cerca de 50 mil millones de objetos. En la Figura 17, se observa que la previsión de objetos conectados en la industria a lo largo de los años será mayor que los conectados para usos médicos o para fines de consumo; lo que plantea a la industria la necesidad de aumentar la seguridad tanto en los objetos IIoT como en los sistemas implementados dentro de ella.

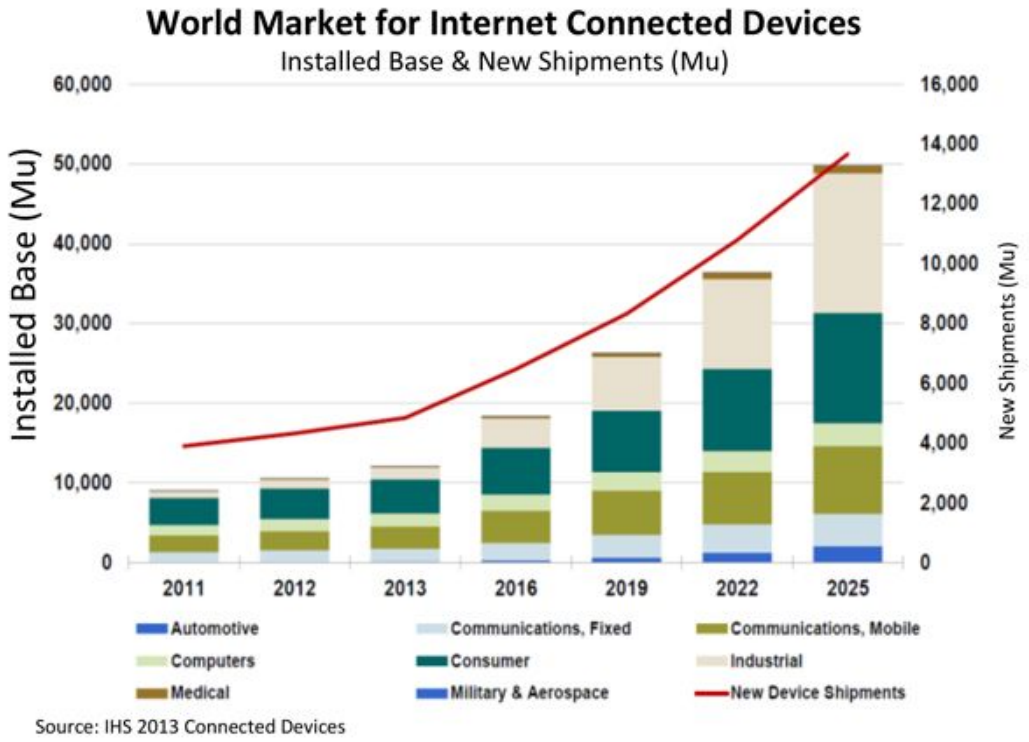


Figura 17: Tendencia de dispositivos conectados a Internet a lo largo de los años

En el apartado anterior se comentaron algunas de las vulnerabilidades que aparecen en los objetos IoT, las cuales son aprovechadas por los ciberdelincuentes para provocar ataques en los objetos y sistemas IoT. Según [17], algunos de estos ataques son:

- Ataque por fuerza bruta, para obtener claves.
- Ataques por denegación de servicio: saturan la red y no permite acceder a los dispositivos.
- Obtención de datos, como hábitos de uso, contraseña de acceso a servicios web, contraseñas de tarjetas, etc.

Para prevenir estos ataques, en [17, 26, 26], se realizan una serie de recomendaciones:

- Actualización y parche, tanto de los objetos como del sistema operativo que se utiliza.
- Autenticación, control de acceso y administración. Hay que evitar las contraseñas predeterminadas y utilizar contraseñas robustas en su lugar, en el acceso a la configuración de los dispositivos IoT.
- Protección de datos almacenados. Controlando qué datos mandamos al fabricante y cuáles no.
- Uso de protocolos seguros. El protocolo de comunicaciones, debe ser el encargado de proporcionar la seguridad en los datos que comparte, asegurando que no se corrompen ni se pierden.

- Bloquear puertos. De este modo se cierran posibles brechas de seguridad dentro del sistema. Hay que dejar abiertos solo los puertos que se emplean para los diferentes servicios utilizados.
- Plan de continuidad de servicio. Tener un dispositivo que almacena en registros lo que sucede, por ejemplo, quien intenta acceder al dispositivo, la hora, el día, la IP de origen, etc.
- Borrado de información antes de desechar el dispositivo.

3 Labview

3.1 Introducción

Como se ha comentado en el capítulo 2, IoT está convirtiéndose en una herramienta importante en la vida cotidiana, y por extensión, sucede lo mismo con IIoT. Tal y como se indica en [11], tanto las antiguas empresas como las de reciente creación, se están transformando para aprovechar la magnitud de oportunidades que ofrece IIoT.

La utilización de la nueva tecnología ya no es una ventaja, sino una necesidad para poder encontrarse entre los mayores competidores en el mercado. Los dispositivos IIoT hacen que las empresas que los utilizan sean más eficientes, abaraten costes y tengan mayor capacidad de gestión de datos obtenidos, haciendo posible que su empresa se integre en la nueva generación de la industria, es decir en la Industria 4.0.

En el mundo industrial, con la aparición de IIoT, se incorpora una red de máquinas y dispositivos con mayor inteligencia que deben ser definidos y controlados por mediación del software.

Una de las mayores dificultades que debe abordarse, es la inclusión de dispositivos y máquinas en el diseño del sistema, dado que, en muchas ocasiones, los equipos utilizan protocolos y desarrollos de prueba diferentes, haciendo que la implementación del sistema sea compleja. Para ello, lo ideal es emplear un software que permita la utilización de diferentes protocolos y así poder homogeneizar todo el diseño, simplificando tiempo y costes.

Una de las empresas que se dedica a este campo, fabricación y venta de dispositivos IIoT y proveedor de software que facilita la implementación de diseños es NI, y uno de estos softwares, es *Labview*.

En los siguientes subapartados, se hará un repaso básico de la herramienta, su metodología de programación y de las características y funciones básicas que se pueden utilizar en ella.

3.2 Qué es *Labview* y para qué sirve

Labview es una plataforma de desarrollo que permite programar en un entorno gráfico, facilitando el acceso a la programación a todos aquellos usuarios que no tienen conocimientos de lenguajes de programación, como C.

Es un software de NI y ellos mismo lo definen en [27] como "*un software de ingeniería diseñado para aplicaciones que requieren pruebas, medidas y control con acceso rápido a información de datos y hardware".*

El objetivo fundamental de *Labview* es conseguir reducir el tiempo de desarrollo del producto final y permitir que profesionales de cualquier campo puedan realizar sus propios diseños sin necesidad de ser expertos en programación.

Labview ofrece la posibilidad de trabajar con diverso hardware: ya sea propio de NI, como de otros fabricantes. Dentro de los hardware de NI podemos encontrar sistemas basados en PC, CompactRIO, PXI, etc. Todos estos sistemas tienen módulos de interfaces de comunicaciones, entradas analógicas y digitales, sensores de temperatura, etc.

Algunos campos de trabajo con *Labview* son:

- Comunicaciones
- Tiempo Real
- Adquisición y tratamiento de imágenes
- FPGAs
- Sincronización de dispositivos
- Etc.

3.3 Metodología de programación en *Labview*

Hay que tener en cuenta que la programación que se realiza en *Labview* es secuencial y por lo tanto es importante poner a la izquierda las entradas y las salidas a la derecha, para conseguir una correcta lectura y depuración posterior.

Si se desea que se ejecuten paralelamente varias tareas dentro de un mismo VI, hay que separar esas tareas y englobarlas en diferentes bucles, de manera que no intercambien datos entre ellos.

La metodología básica de programación de los VIs es la siguiente:

- Adquisición

  Se obtienen los datos con lo que se va a trabajar con la ayuda de los controladores que se pueden encontrar en la página de National Instrument [28]. De tal manera que se puede configurar las entradas y el tipo de datos que se desean adquirir para un hardware especifico, tanto propio de NI como de otros fabricantes.

  Es muy común emplear VI Express en esta parte del programa, agilizando así la tarea de configuración del dispositivo; o generando señales simuladas para la primera parte de programación de un proyecto, donde hay que depurar y realizar pruebas antes de conectar el hardware.

- Análisis

Para analizar los datos, existen múltiples controles en la biblioteca de *Labview*. Tambien se pueden emplear VI Express.

- Presentación.

  Para la presentación de los datos, se tiene infinidad de indicadores como:

  - Numeric Indicators,
  - Waveform Chart or Waveform Graph,
  - XY Graph.

  Pero si se quiere algo más específico como crear un texto para mostrar posteriormente, o registrar los datos en un fichero o generar una señal para atacar a un hardware, es recomendable usar VI Express.

Para mayor detalle sobre la programación con Labview, dirigirse al Anexo de este TFG.

4 Protocolo de comunicaciones MQTT

4.1 Introducción

El Transporte de Telemetría de Cola de Mensajes, MQTT, fue inventado en 1999 por Andy Stanford-Clark (IBM) y Arlen Nipper (Arcom, ahora Cirrus Link), buscando un protocolo que ocasionará una pérdida mínima de batería y un ancho de banda mínimo.

Según [29], los requisitos para el desarrollo del protocolo son:

- Implementación simple,
- Entrega de datos de calidad de servicio,
- Ligero y ancho de banda eficiente,
- No es necesario que conozca los datos que maneja,
- Debe poder mantener una sesión continua.

El resultado es un protocolo de código abierto que se desarrolló y optimizó para dispositivos restringidos y redes de bajo ancho de banda, alta latencia o poco confiables. [30]

Es un protocolo que se engloba en la categoría publicador/suscriptor que es extremadamente ligero y permite la interconexión de dispositivos pequeños a redes con un ancho de banda mínimo.

Tiene la finalidad de minimizar los requerimientos de recursos del dispositivo y asegurar la confiabilidad y cierto grado de seguridad de entrega con calidad de servicio.

Está orientada a grandes redes de dispositivos pequeños que necesitan la supervisión o el control de un servidor en Internet. MQTT es simple, bastante fácil de implementar y ofrece pocas opciones de control.

4.2 Conocimientos básicos

Como se ha detallado en el capítulo 2.6, MQTT es un protocolo que se engloba dentro del modelo publicación/suscripción.

Dentro de este modelo de comunicación, MQTT está preparado para funcionar filtrando los mensajes por “temas” o “*Topics*”; de manera que el Broker al recibir el mensaje determina a través del *Topic* a qué suscriptor debe mandar el mensaje recibido.

A continuación, se describen de manera sencilla algunas de las funcionalidades y mensajes del protocolo.

4.2.1 Conexión MQTT

De [31], se extraen las definiciones de Cliente y Broker MQTT, así como el procedimiento para establecer la conexión entre ambos.

Se denomina Cliente MQTT a aquel dispositivo que puede publicar y/o suscribirse a un *Topic* y se conecta con un *Broker* MQTT a través de una red ejecutando una biblioteca MQTT.

Se denomina *Broker* MQTT a aquel intermediario responsable de recibir todos los mensajes, filtrarlos y reenviarlos a los clientes suscritos a ese *Topic*. El *Broker* es responsable de la autenticación y autorización de los clientes dentro de comunicación y de garantizar el mantenimiento de la sesión de todos los clientes persistentes.

Como se ha explicado en 2.6, MQTT es un protocolo que trabaja sobre TCP/IP, por lo tanto, los elementos que interactúan con este protocolo deben tener una pila TCP/IP.

Tal y como se puede apreciar en la Figura 18, el cliente envía el mensaje CONNECT y el *Broker* debe responder con el mensaje CONNACK.

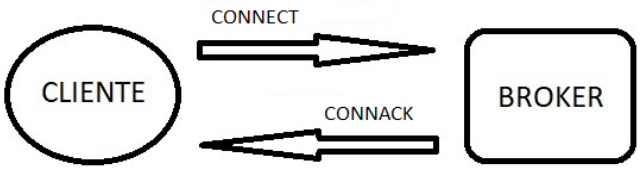


Figura 18: Ejemplo de mensajes de conexión

El *Broker* debe mantener la sesión abierta mientras el cliente no envíe un mensaje DISCONNECT o se interrumpa la conexión.

El mensaje CONNECT debe tener un formato correcto para que el *Broker* responda con el mensaje CONNACK; de este modo se pretende disuadir a los dispositivos maliciosos de ralentizar al *Broker*.

Algunas de las opciones más importantes que debe contener el mensaje CONNECT son:

1) ClientID: Identifica a cada cliente, siendo único para cada conjunto cliente-*Broker*.
2) CleanSession: Indica si se desea establecer una sesión persistente.
   a) Si es “falso” se establece una sesión persistente y el *Broker* debe almacenar todas las suscripciones para el cliente y todos los mensajes perdidos para el cliente que se suscribe con QoS 1 ó 2.

b) Si es “true” la sesión no es persistente y no es necesario almacenar nada para el cliente y se elimina la información de una sesión persistente anterior.

3) Usuario y Contraseña: Se puede emplear opcionalmente para la autenticación y autorización de un cliente. Para emplear esta opción se recomienda el uso de un certificado SSL.
4) Will Will message: Notifica a otros clientes cuando un cliente se desconecta. Es como un mensaje de última voluntad. Cuando el cliente se desconecta sin desearlo (falta de batería, algún error, etc.), el *Broker* envía este mensaje a los clientes suscritos en nombre del que se desconectó.
5) Keep Alive: Es un intervalo de tiempo en segundos que especifica el cliente al *Broker*, definiendo el intervalo de tiempo máximo que él puede existir sin comunicación entre ellos. El cliente debe enviar mensajes PINGREQ al *Broker* periódicamente respondiendo este con un mensaje PINGRESP.

Por su parte el mensaje CONNACK debe contener:

1) SessionPresent: Indica al cliente si ya se mantiene una sesión persistente disponible de interacciones anteriores con él.
2) ReturnCode: Indica si el intento de conexión se ha realizado con éxito.
   1. **00** : conexión aceptada
   2. **1** : conexión rechazada, versión de protocolo errónea
   3. **2** : conexión rechazada, identificador rechazado
   4. **3** : conexión rechazada, servidor no disponible
   5. **44** : conexión rechazada, nombre de usuario o contraseña incorrecta
   6. **55** : conexión rechazada, no autorizada

4.2.2 Publicación, suscripción y cancelación de suscripción

En [32], se explica los mensajes que intervienen en la publicación, suscripción y cancelación de suscripción.

En la publicación, se intercambia el mensaje PUBLISH entre el cliente y el *Broker* y este y los suscriptores, tal y como se muestra en la Figura 19.

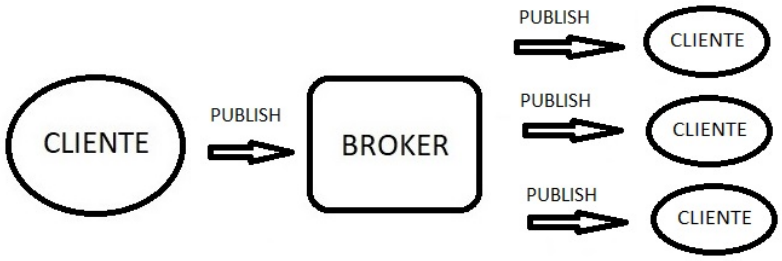


Figura 19: Ejemplo publicación MQTT

Cada mensaje debe contener como mínimo, el *Topic* empleado por el intermediario para reenviar el mensaje, el mensaje a enviar y la calidad de servicio.

El mensaje enviado, puede enviarse entre otros, como dato binario, dato de texto, XML o JSON.

El mensaje PUBLISH debe contener:

1) TopicName: Es una cadena estructurada jerárquicamente con barras diagonales como delimitadores.
2) QoS: Indica el nivel de calidad de servicio del mensaje. (0, 1 ó 2)
3) RetainFlag: Un flag que define si el intermediario guarda el mensaje como el ultimo valor valido conocido para un *Topic* específico. Si está activo, el *Broker* envía el mensaje retenido a los nuevos clientes suscritos para poder conocer el estado del *Topic*.
4) Payload: Contenido del mensaje. Es posible enviar todo tipo de datos.
5) PacketID: Identifica un mensaje entre cliente y *Broker*. Solo se emplea en QoS mayores de cero.
6) Dupflag: Indica si el mensaje es duplicado. Solo se emplea en QoS mayores de cero.

En lo que respecta a la suscripción, para poder recibir los mensajes publicados por otros clientes, un cliente cualquiera debe suscribirse a un *Topic*. Es por ello, que el cliente que desea suscribirse a un *Topic*, envía un mensaje SUBSCRIBE al *Broker*.

El Mensaje SUSBCRIBE contiene:

1) PacketID: Identifica un mensaje entre cliente y *Broker*. Solo se emplea en QoS mayores de cero.
2) Lista de suscripciones: Formado por QoS y TopicName. Pueden existir tantos conjuntos de datos como suscripciones se deseen.
    a) QoS: Indica el nivel de calidad de servicio del mensaje. (0, 1 ó 2)
    b) TopicName: Es una cadena estructurada jerárquicamente con barras diagonales como delimitadores.

Para confirmar la suscripción, el *Broker* devuelve al cliente el mensaje SUBACK, este mensaje contiene:

1) PacketID: Identifica un mensaje entre cliente y *Broker*. Solo se emplea en QoS mayores de cero.
2) ReturnCode: Se envía un código por cada suscripción enviada en el mensaje SUBSCRIBE. Los códigos que pueden aparecer son:
   a) **00** : éxito: QoS 0
   b) **1** : éxito: QoS 1
   c) **2** : éxito: QoS 2
   d) **128** : Fracaso

Un ejemplo de intercambio de mensajes de suscripción se muestra en la Figura 20.

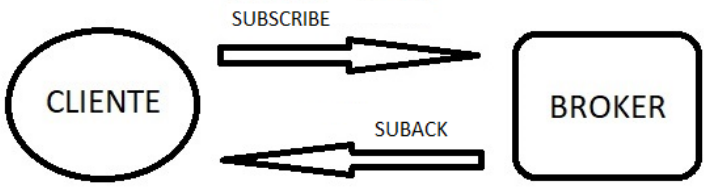


Figura 20: Ejemplo de suscripción MQTT

De manera análoga, cuando un cliente quiere dejar de recibir mensajes de un *Topic*, emplea el mensaje UNSUBCRIBE enviando:

1) PacketID: Identifica un mensaje entre cliente y *Broker*. Solo se emplea en QoS mayores de cero.
2) Lista de *Topics*: Pueden existir tantos *Topics* como suscripciones se deseen anular.

Para verificar la cancelación de suscripción, el *Broker*, envía el mensaje UNSUBACK. Este mensaje solo contiene el identificado de paquete.

Un ejemplo de intercambio de mensajes de cancelación de suscripción se muestra en la Figura 21.

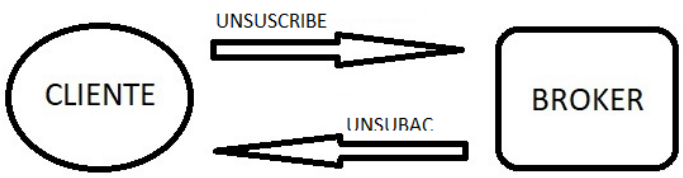


Figura 21: Ejemplo de cancelación de suscripción MQTT

4.2.3 Calidad de Servicio, QoS

Para manejarse dentro del modelo de comunicación, MQTT evalúa tres niveles de QoS, tal y como se deduce de [33]:

[1] QoS (0): A lo más una vez.

El mensaje se transmite una sola vez. Es el mínimo nivel de QoS, no se garantiza la entrega del mensaje. No se solicita acuse de recibo y el remitente no almacena ni reenvía el mensaje.

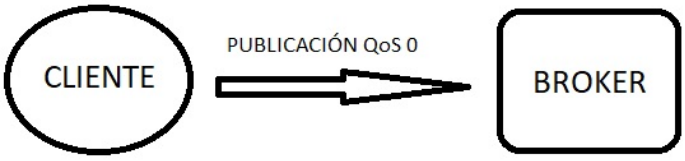


Figura 22: Ejemplo MQTT con nivel QoS 0

[2] QoS (1): Al menos una vez

Se garantiza que el mensaje se entregue por lo menos una vez al receptor. El publicador almacena el mensaje hasta recibir un paquete PUBACK del receptor que se emplea como acuse de recibo. Se puede enviar el mismo mensaje varias veces, hasta que el receptor recibe el mensaje.

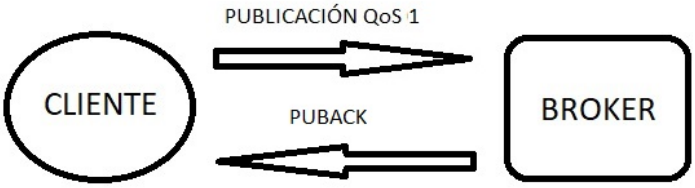


Figura 23: Ejemplo MQTT con nivel QoS 1

En este caso, el paquete PUBLISH usa un identificador que será empleado por el receptor para devolver el mensaje PUBACK. Si el remitente no recibe el mensaje PUBACK en un tiempo razonable, vuelve a mandar el mensaje PUBLISH.

Si el cliente de publicación envía el mensaje de nuevo, pone a 1 el indicador de duplicado, DUP. En este caso, esta información es trasparente para el receptor del mensaje.

[3] QoS (2): Exactamente una vez

Este nivel garantiza la entrega del mensaje al destinatario una sola vez. Es el nivel de calidad más lento y seguro. Por cada mensaje hay un intercambio de cuatro paquetes de datos entre el emisor y el receptor del mensaje. Ambas partes emplean el mismo identificador en los cuatro mensajes intercambiados.

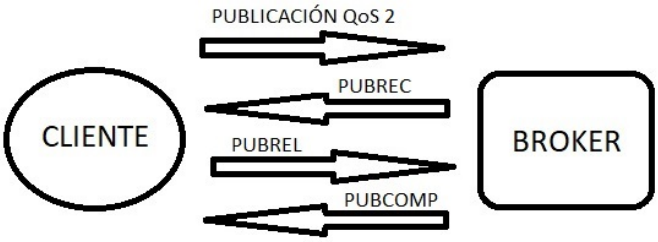


Figura 24: Ejemplo MQTT con nivel QoS 2

En este caso, cuando el receptor recibe un mensaje con QoS 2, envía un mensaje PUBREC para reconocer el paquete PUBLISH. Y si el emisor no recibe dicho paquete, vuelve a enviar el PUBLISH con el indicador DUP hasta recibir el mensaje PUBREC.

Si el emisor recibe el paquete PUBREC, contesta al receptor con el paquete PUBREL y desecha el paquete PUBLISH que tenía almacenado.

Después de recibir el paquete PUBREL, el receptor puede descartar los mensajes almacenados y responder con un PUBCOMP. Cuando el emisor recibe PUBCOMP, se libera el indicador, ambas partes eliminan las copias almacenadas y el indicador queda a disposición de poder volver a ser utilizado en un nuevo mensaje.

Hay que tener en cuenta algunos aspectos importantes a la hora de trabajar con los QoS:

1) Tal y como se muestra en la Figura 25, aunque el cliente inicie la comunicación del mensaje con PUBLISH QoS 2, el *Broker*, no tiene por qué reenviar el mensaje con esa misma calidad de servicio. De hecho, el *Broker* enviará los mensajes a los suscriptores del mismo modo al que se suscriben al *Topic*.

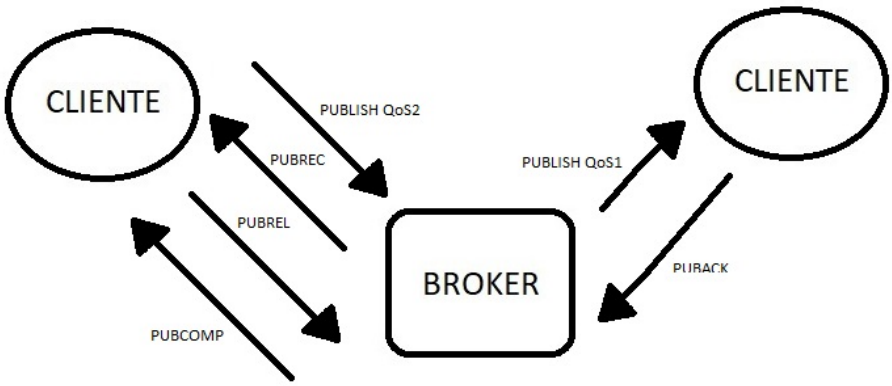


Figura 25: Ejemplo de uso QoS

2) No en todos los casos es necesario trabajar con una QoS elevada. Siempre es importante evaluar cada caso por separado para decidir qué QoS implementar. Algunos ejemplos de utilización son:
   a) QoS 0, cuando:
      i) Conexión estable entre emisor y receptor.
      ii) No importa que se pierdan algunos mensajes ocasionalmente. Dado que se envían a intervalos cortos.
      iii) No es necesario hacer cola de mensajes.
   b) QoS 1, cuando:
      i) Es necesaria la recepción de todos los mensajes y se deben pueden manejar mensajes duplicados.
      ii) En más rápido que QoS 2 y requiere menos procesamiento.
   c) QoS 2, cuando:
      i) Es fundamental para el caso de uso recibir todos los mensajes una sola vez. Lo malo de implementar este caso es que la red se sobrecarga y en consecuencia el tiempo de completarse el intercambio de un mensaje es más elevado.

4.2.4 Topics

En [34], se aclara el concepto de *Topic* y hablan de estrategias de utilización de los mismos.

Un *Topic* es una cadena de palabras que el *Broker* emplea para filtrar los mensajes recibidos y enviarlos a los suscriptores.

Un *Topic* consta de uno o más niveles y cada nivel está separado por una barra diagonal. Se distinguen entre mayúsculas y minúsculas. Cada *Topic* debe tener al menos un carácter.

Se pueden emplear comodines al suscribirse a múltiples *Topics* simultáneamente. Solo es válido para suscribirse, no para publicar. Existen dos tipos de comodines:

1) Single_Level (+): Reemplaza a un nivel del *Topic*.
2) Multi_Level (#): Reemplaza muchos niveles de *Topics*. Debe colocarse como el ultimo carácter del *Topic*.

Por ejemplo, si tenemos una estructura jerárquica como la mostrada en la Figura 26.

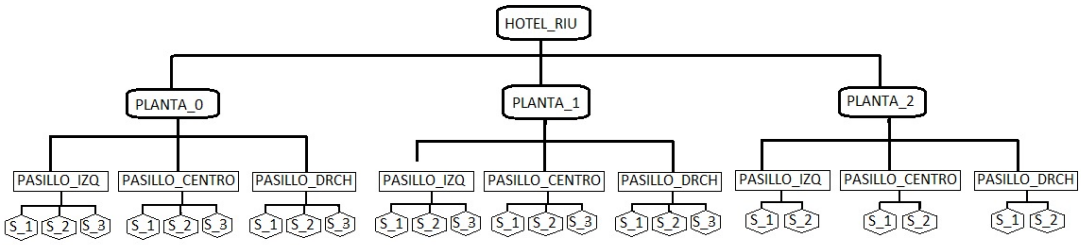


Figura 26: Ejemplo de estructura jerárquica

Podríamos encontrar *Topics* completos como:

- HOTEL_RIU/PLANTA_0/PASILLO_CENTRO/S_1
- HOTEL_RIU/PLANTA_2/PASILLO_DRCH/S_2
- HOTEL_RIU/PLANTA_1/PASILLO_IZQ/S_3

Empleando comodines nos podemos encontrar con:

- HOTEL_RIU/+/PASILLO_CENTRO/S_1
- HOTEL_RIU/PLANTA_1/#

En el caso del comodín Single_Level: el cliente se suscribe a todos los sensores que hay en los PASILLO_CENTRO de todas las plantas.

En el caso del comodín Multi_Level: el cliente se suscribe a todos los sensores que cuelgan de todos los pasillos de la PLANTA_1.

4.3 Especificación MQTT

En abril de 2014 OASIS fue la encargada de realizar la especificación del protocolo de comunicaciones MQTT, cuya versión actual es 3.1.1.

En junio de 2016, el protocolo MQTT se estandariza y se convierte en una norma ISO (ISO /IEC 20922) [35].

En ella se describe en detalle:

1) El formato de los paquetes,
2) el tipo de paquetes que se transmiten,
3) el comportamiento de operación,
4) la seguridad,
5) el uso de WebSocket como red de transporte.

En el apartado anterior se han comentado de manera básica los mensajes que se transmiten para el correcto funcionamiento de MQTT, sin entrar en detalle en el contenido de los bits de cada mensaje, ni la respuesta de los paquetes.

Si se desea tener un conocimiento más exhaustivo del protocolo, en [36] se profundiza más en cada uno de los mensajes, aclarando cualquier duda que pueda surgir entorno a los bytes de las cabeceras y la respuesta esperada a cada mensaje para la versión 3.1.1.

En marzo de 2019, apareció la versión 5.0 del estándar de MQTT [37]. Esta nueva versión introduce mejoras interesantes, pero aún debe que minimizar las incompatibilidades con las versiones anteriores.

En [38], se hace un resumen de los aspectos más destacados de esta nueva versión, y entre ellos se puede encontrar:

1) Mejor informe de errores:

   Se amplían las posibilidades de error agregando (opcionalmente) códigos de motivos a las respuestas de los mensajes PUBACK y PUBREC, PUBREC, CONNACK, UNSUBACK, DISCONNECT, SUBACK y AUTH.

2) Suscripciones compartidas:

   Para *Topics* cuya tasa de suscripción es elevada, las suscripciones compartidas ayudan a equilibrar la carga de los mensajes en varios clientes suscritos.

3) Propiedades de los mensajes:

   Se introducen metadatos en la cabecera del mensaje para uso del usuario.

4) Caducidad del mensaje:

   Es una opción que posibilita el descarte del mensaje, transcurrido un tiempo definido sin que se haya producido la entrega del mismo.

5) Caducidad de la sesión:

   Si tras una desconexión del cliente, este no se vuelve a conectar transcurrido un tiempo definido por el usuario, se pueden descargar los mensajes almacenados en la sesión anterior sin necesidad de limpiar la misma.

6) Alias del *Topic*:

   Permite sustituir *Topics* largos por números enteros, reduciendo así el tamaño de la sobrecarga del mensaje.

7) Descubrimiento de funciones permitidas:

   En la conexión se puede definir un límite máximo de tamaño de paquete y número de mensajes que pueden ser transmitidos para informar de lo que está sucediendo.

Algunos *Brokers* y Clientes, ya están trabajando en pruebas con las nuevas funcionalidades incorporadas en esta nueva versión.

4.4 Brokers MQTT

En la actualidad hay varias opciones a la hora de usar un *Broker* MQTT, se puede optar por emplear un *Broker* que se encuentra en la nube o instalar un *Broker* en un PC o máquina virtual y configurarlo para trabajar en local.

Si se opta por un *Broker* que se encuentra en la nube, existen dos opciones:

1) *Broker* MQTT privados

   Dentro de la amplia gama de opciones, hay tres que son los más utilizados. Solo pueden publicar y suscribirse a este tipo de *Broker* los dispositivos que se registran en el *Broker*.

   Mientras algunos como Azure permite 8000 mensajes/día de manera gratuita, CloudMQTT permiten 5 conexiones/día y una velocidad de 10Kbits/s y AWS permite 250000 mensajes/mes. La gran diferencia entre ellos es la configurabilidad que presenta CloudMQTT frente a los otros dos, permitiendo al usuario configurar los puestos de conexión de los dispositivos y el *Broker*.

   En la Figura 27 se muestra una tabla con las diferencias principales entre estos tres *Brokers* privados.

Broker	URL	Puerto Regular / TCP	Puerto SSL / TLS	Puerto Web Socket	Mens Retain	Sesión Persist	Nivel QoS	Free Limits
Azure	https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-mqtt-support	N/A	8883	443	NO	SI	0, 1	8000 m/día
AWS	https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html	N/A	8883	443	NO	SI	0, 1	250,000 m/mes
CloudMQTT	https://www.cloudmqtt.com/plans.html	Conf Port	Conf Port	Conf Port	N/A	SI	0, 1, 2	5 Con/día & 10 Kbit/s

Figura 27: Brokers MQTT Privados [39]

2) *Broker* MQTT Públicos

En un *Broker* público, cualquier dispositivo puede publicar y suscribirse a un *Topic* que se encuentre en el *Broker*; no hay privacidad. Nunca deben ser empleados en producción. Son muy útiles para aprender a utilizar el protocolo de comunicaciones.

En la Figura 28 se muestran las diferencias entre algunos de los más conocidos, también encontramos el enlace a cada uno de los *Brokers*, donde se especifica más detalladamente sus principales características y especificaciones.

Broker	*Broker* Address	Port TCP	Port SSL/ TLS	Port Web Socket	Mens Retain	Sesion Persist	Registro	Link
Eclipse	iot.eclipse.org	1883	N/A	80, 443	SI	SI	NO	http://iot.eclipse.org/sandbox.html
Mosquitto	test.mosquitto.org	1883	8883, 8884	80	SI	SI	NO	http://test.mosquitto.org/
HiveMQ	broker.hivemq.com	1883	N/A	8000	SI	SI	NO	https://www.hivemq.com/try-out/
Flespi	mqtt.flespi.io	1883	8883	80, 443	SI	SI	SI	https://flespi.com/mqtt-broker
Dioty	mqtt.dioty.co	1883	8883	8080, 8880	SI	SI	SI	http://www.dioty.co/
Fluux	mqtt.fluux.io	1883	8883	N/A	N/A	N/A	NO	https://fluux.io/

Figura 28: Brokers MQTT Públicos [39]

Si se opta por trabajar en local instalando un *Broker* en un PC o máquina virtual, primero se debe escoger el servidor con el que se quiere trabajar.

La mayoría de los *Brokers* tienen sus propias especificaciones y los requerimientos mínimos que debe tener la máquina donde se instalará el *Broker* para el correcto funcionamiento del mismo.

Dentro de los más conocidos, se encuentran: HiveMQ, Mosquitto, Eclipse Mosquitto.

Todos ellos se pueden instalar sobre MacOS, Windows, y distribuciones Linux. En sus páginas web, se encuentra el paquete de datos a descargar y una explicación de cómo debe realizarse la instalación.

4.5 Clientes MQTT

Como se ha comentado en apartados y capítulos anteriores, cualquier dispositivo que se pueda conectar a una red y que pueda ejecutar una librería MQTT es un cliente potencial para el protocolo.

Es por lo tanto importante conocer algunas de las librerías necesarias para que cualquier dispositivo pueda conectarse a una red y trabajar con el protocolo.

También se puede considerar cliente a una aplicación que se encuentre instalada en un dispositivo, en este caso estamos hablando de simuladores de clientes o aplicaciones cliente.

4.5.1 Librerías MQTT Client

Todas estas librerías proveen al dispositivo de las herramientas necesarias para conectarse con un *Broker* MQTT y poder publicar mensajes y suscribirse a *Topics* recibiendo así los mensajes publicados en ellos.

1) Paho-mqtt

Tiene librerías para casi todos los lenguajes de programación. Las funciones más comunes en todos los lenguajes son:

- Client(): genera un cliente con el cuál conectarse al *Broker* MQTT.
- Connect(): permite la conexión con el *Broker*.
- Disconnect(): permite la desconexión del cliente con el *Broker*.
- Publish(): Envía un mensaje desde el cliente al *Broker*.
- Subscribe(): permite suscribir el cliente a un *Topic*.

En [40] se encuentran las librerías y las propiedades para los diferentes lenguajes en función del sistema operativo donde se pretende instalar el cliente. Se pueden encontrar librerías para:

- Python
- C para Posix and Windows 3.1.1.
- C++ para Posix and Windows
- C/C++ para embebidos
- Java
- Android Service

- JavaScript

Todas estas librerías se pueden descargar gratuitamente desde [41].

2) Libmosquitto

Librería en lenguaje C que implementa la versión 3.1.1 del protocolo. Emplea funciones como:

- Mosquitto_connect (): permite la conexión con el *Broker*.
- Mosquitto_disconnect(): permite la desconexión con el *Broker*.
- Mosquitto_publish(): permite la publicación de un mensaje en un *Topic* determinado.
- Mosquitto_subscribe (): permite suscribirse a un *Topic* determinado.

Muchas más funciones empleadas en esta librería se describen en [42].

3) AWS-IoT-RESTful

Librería para *Labview*, con la cual se puede conectar al cliente con el *Broker* AWS de Amazon. Esta librería contiene un proyecto en el que se emplean VIs que implementan certificación SSL en la conexión para dotar de seguridad a la misma.

Esta librería se puede descargar de [43].

4) LVMQTT

Librería para *Labview* de daq.io en la cual se engloban los VIs necesarios para la creación de un cliente en *Labview* que es capaz de conectarse a un *Broker* y comunicarse a través de MQTT, tanto publicando como suscribiéndose a *Topics*.

Esta librería no está dotada de seguridad con certificación SSL. Se puede descargar en [44]

4.5.2 Simuladores de clientes: Aplicaciones

Existen en el mercado aplicaciones que simulan un dispositivo y a través de las cuales se puede acceder a un *Broker* MQTT y suscribirse y publicar en los *Topics* que se desee.

Estas aplicaciones son muy útiles para familiarizarse con el protocolo y comprender su funcionamiento, así como para los desarrolladores de controles y sistemas IoT, dado que con ellas pueden depurar sus trabajos.

En el mercado se pueden encontrar algunas de estas aplicaciones. Según [45], las cinco mejores son:

1) **MQTT.fx** [46]

Disponible para los SSOO más comunes: MacOSX, Linux, Win. Se implementa con JavaFX.

Además de las funciones de publicación y suscripción, también permite soporte de *Topics* $ SYS (*Topics* internos de configuración del *Broker*) y conectarse a diferentes *Brokers*.

Como inconveniente tiene que las suscripciones y las publicaciones están en pestañas separadas, por lo tanto, la depuración se hace un poco más incómoda.

En la Figura 29 se observa un ejemplo del entorno de esta aplicación.

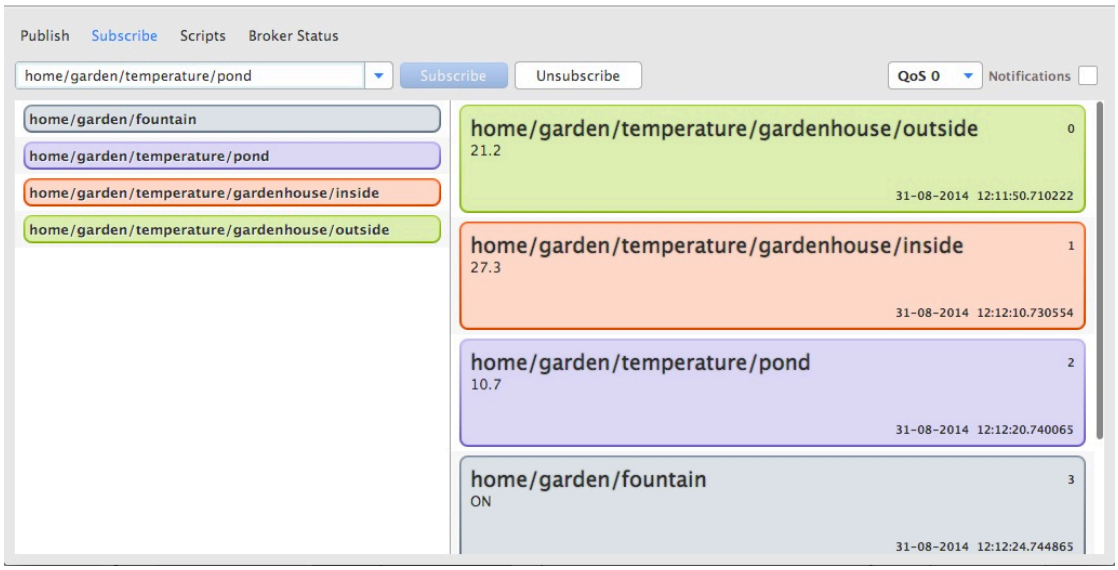


Figura 29: Aplicación MQTT.fx [46]

2) mqtt-spy

Está basado en Java 8, se ejecuta sobre Java 8 y JavaFX. Forma parte de los proyectos Eclipse IoT y Eclipse Paho.

Es muy visual y facilita poder trabajar con la publicación y la suscripción en una misma ventana. Separa por pestañas los diferentes *Brokers*, y las diferentes suscripciones, mostrándolas con diferentes colores. También tiene un área donde poder ver todos los mensajes que llegan con cada *Topic* en el color correspondiente a la suscripción. De esta forma se pueden ver de un vistazo todos los mensajes publicados en todos los *Topics* suscritos.

Almacena los mensajes de cada *Topic* en archivos diferentes, permitiendo analizarlo más tarde más exhaustivamente.

En la Figura 30 se observa un ejemplo del entorno de esta aplicación.

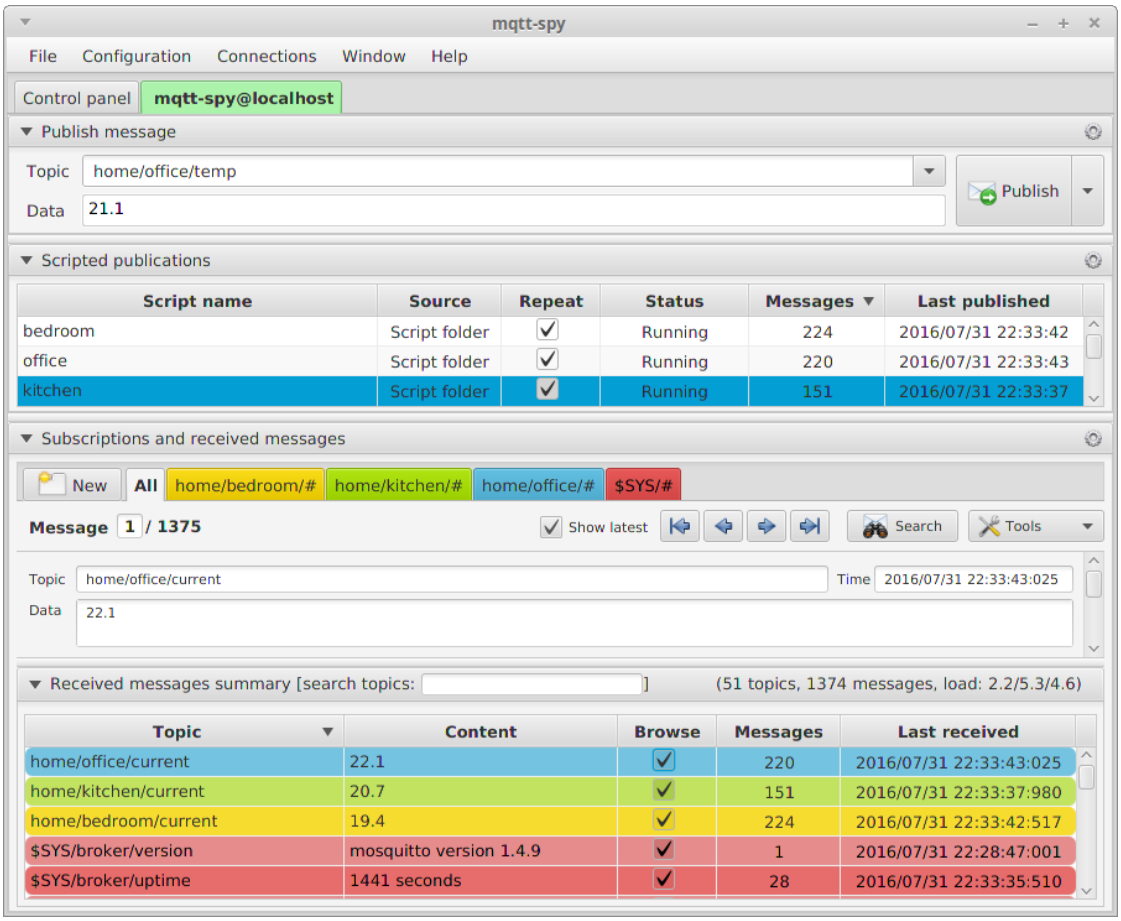


Figura 30: Aplicación mqtt-spy [45]

3) **MQTT Inspector** [47]

Es de uso exclusivo en dispositivos con iOS. Es muy útil para usuarios avanzados de MQTT. No es una aplicación gratuita.

Permite configurar plantillas de publicación y suscripción, filtrar por *Topics*, atributos o carga útil; también permite agrupar mensajes.

Como en el caso anterior, diferencia los mensajes de diferentes *Topics* con colores para mostrar todos los mensajes en una única ventana.

En la Figura 31 se observa un ejemplo del entorno de esta aplicación.

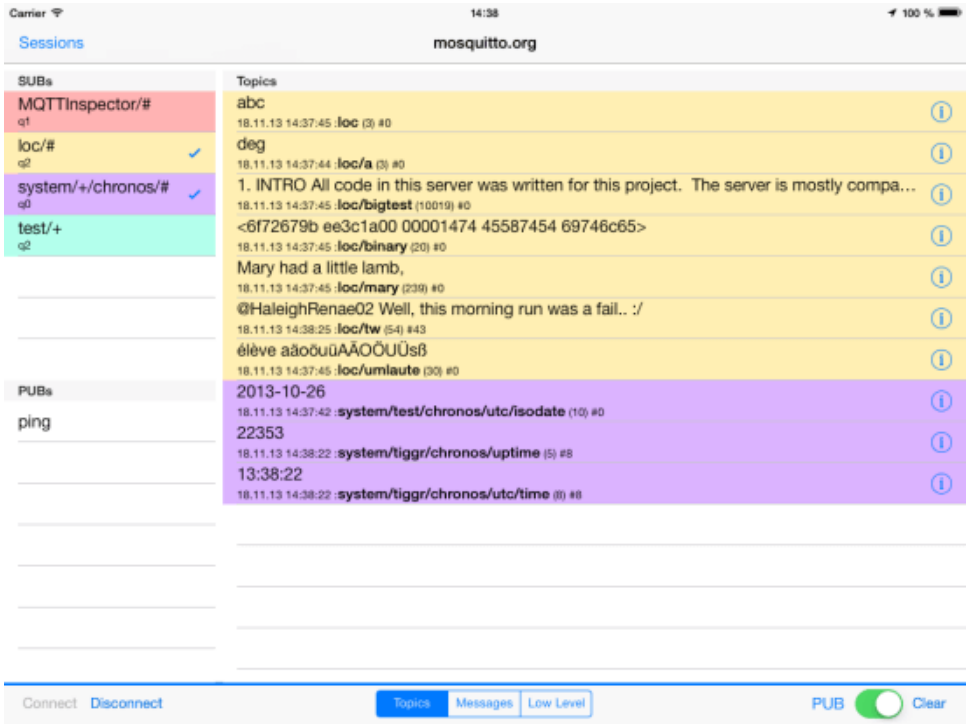


Figura 31: Aplicación MQTT Inspector [47]

4) **MyMQTT** [48]

Es de uso exclusivo para dispositivos Android. Además de las utilidades de publicación y suscripción intrínsecas al protocolo, permite almacenar los mensajes en el teléfono.

En la Figura 32 se observa un ejemplo del entorno de esta aplicación.

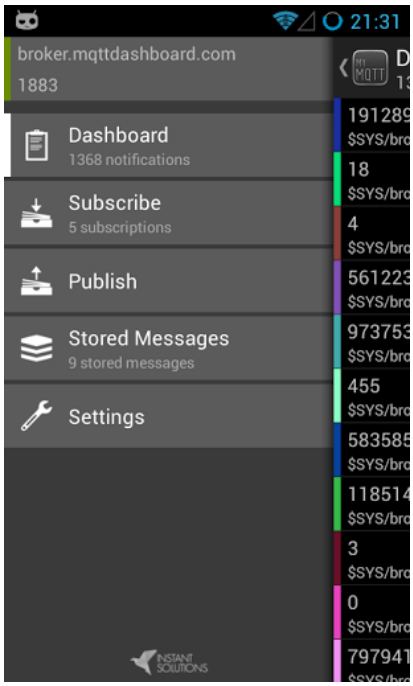


Figura 32: Aplicación MyMQTT [48]

5) HiveMQ WebSocket Client

Herramienta Web que no contempla la sesión persistente con el *Broker*. Es muy útil para trabajar con MQTT a través de WebSocket. Muestra los mensajes coloreados para diferenciar los *Topics* en el área de mensajes.

En la Figura 33 se observa un ejemplo del entorno de esta aplicación.

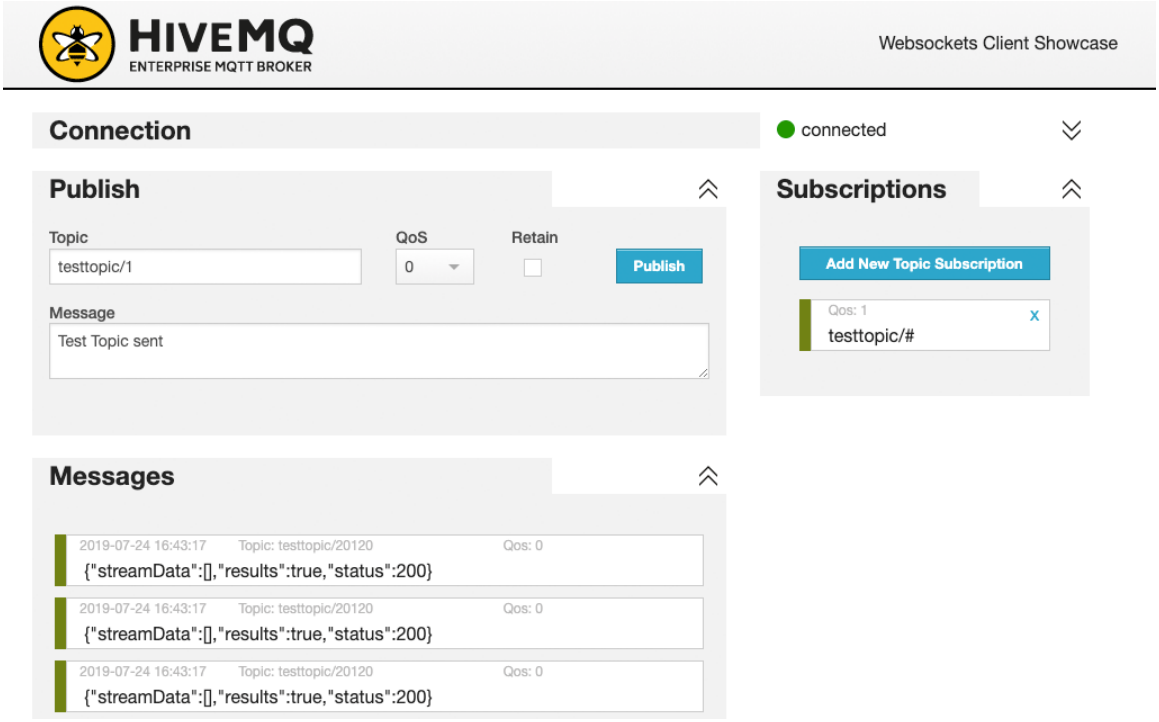


Figura 33: Aplicación HiveMQ WebSocket Client [45]

Además de las aplicaciones mencionadas hasta ahora, en el mercado existen otras aplicaciones, como MQTTbox.

6) **MQTTbox** [49]

Es una aplicación muy cómoda dado que separa en pestañas los diferentes *Brokers* a los que se puede conectar. Dentro de cada pestaña se pueden ver las diferentes publicaciones y suscripciones en pequeñas ventanas.

Soporta las suscripciones y publicaciones con comodines, autenticaciones a través de usuario y contraseña, conexiones con: Keep Alive, Last Will, sesiones persistentes, mensajes retenidos y los tres niveles de calidad de servicio.

No implementa seguridad con certificación SSL, ni *scripting*, pero es visual y fácil de manejar.

En la Figura 34 se observa un ejemplo del entorno de esta aplicación.

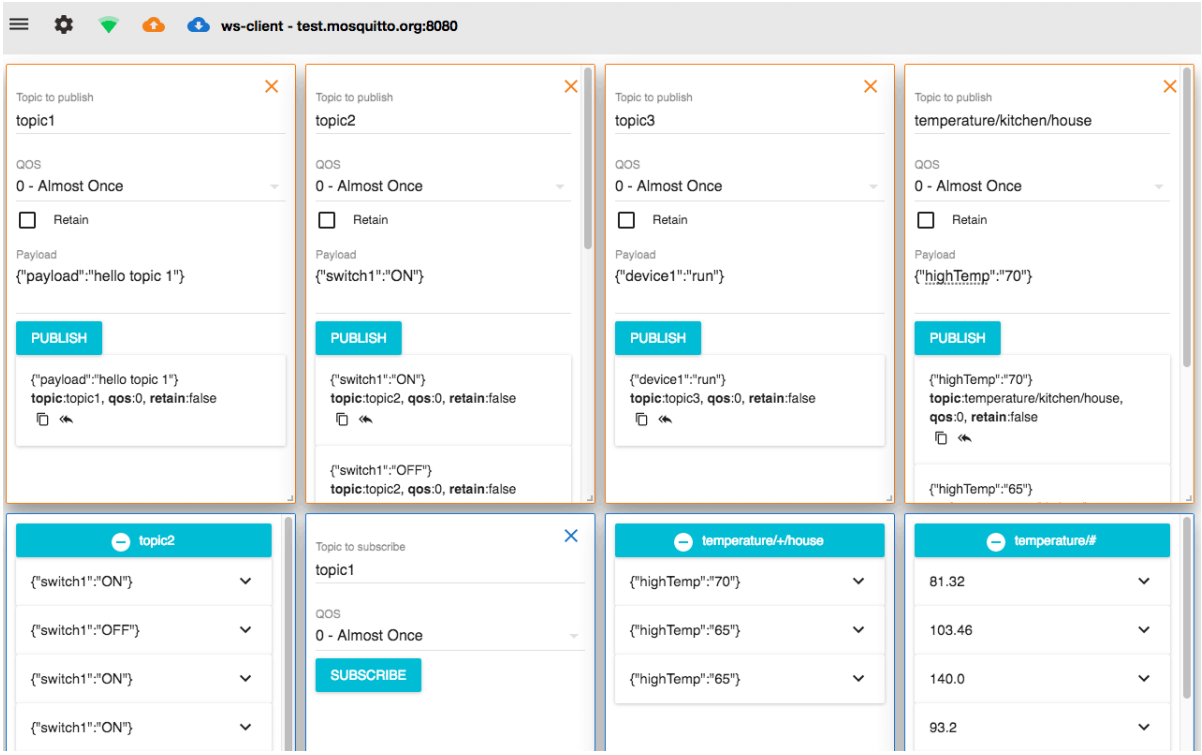


Figura 34: Aplicación MQTTbox [49]

5 Librería MQTT para *Labview*: LVMQTT

En el capítulo anterior, se ha comentado que las librerías MQTT son necesarias para que un dispositivo pueda convertirse en un potencial cliente MQTT.

Entre otras librerías, se mencionan dos librerías para poder utilizarlas en el entorno de desarrollo *Labview*. Una de ellas implementa una mayor seguridad trabajando con certificados SSL, mientras que la otra, más básica, no implementa dicha seguridad, dejando la seguridad de la comunicación, a la inherente en la pila TCP/IP.

Para el desarrollo de este TFG y teniendo en cuenta que no es necesario implementar seguridad en el mismo, dado que el rango de este TFG es la interoperabilidad del software de NI, *Labview*, con el protocolo de comunicaciones MQTT, se ha decidido emplear la librería básica que no introduce seguridad SSL.

Esta librería se puede descargar de manera gratuita en GitHub [44], y va a ser la base utilizada para el desarrollo de la parte práctica del TFG.

En dicha librería se encuentra un conjunto de VI y subVI necesarios para la comunicación MQTT. Todos ellos tienen gestión de errores, presentando entradas y salidas de error. Si a la entrada del VI se ha producido un error, este se propaga a la salida de manera inmediata.

Adjunto a este TFG se entrega la librería para su correcta visualización. En los apartados siguientes se intentará hacer mención a las partes más importantes de cada uno de los VI necesarios para el correcto funcionamiento del protocolo.

5.1 MQTT_Connect.vi

Este VI es el encargado de establecer la conexión entre el *Broker* y el Cliente.

En la Figura 35, se muestra el icono del VI y el conjunto de parámetros de entrada y las salidas necesarias para su posterior procesamiento, destacando el ID de conexión (connection ID out) y el tipo de conexión (Connection Status).

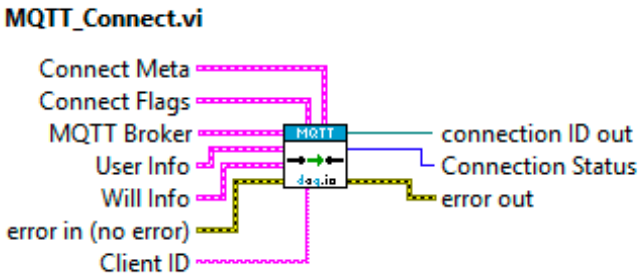


Figura 35: MQTT_Connect.vi

Analizando el VI, se observa en el PF, como se muestra en la Figura 36, que los parámetros de entrada necesarios para establecer la conexión son:

[1] MQTT *Broker*: Dirección IP y puerto al que debe enviar la solicitud de conexión.

[2] ClientID: Nombre con el que se quiere dar a conocer el cliente.

[3] Connect Meta: Parámetros de configuración del cliente. Se especifica el tiempo de Keep Alive, el QoS con el que va a trabajar el cliente y si acepta con trabajar con mensajes duplicados y con mensajes retenidos.

[4] User Info: Informa de un nombre un nombre de usuario y una contraseña para comunicarse con el *Broker*.

[5] Will Info: Informa del mensaje y el *Topic* a enviar si se pierde la conexión.

[6] Connect Flag: Con estos flags, indica al *Broker* si va a emplear los datos mencionados anteriormente (usuario y contraseña, will info) y si quiere emplear una sesión persistente o no.

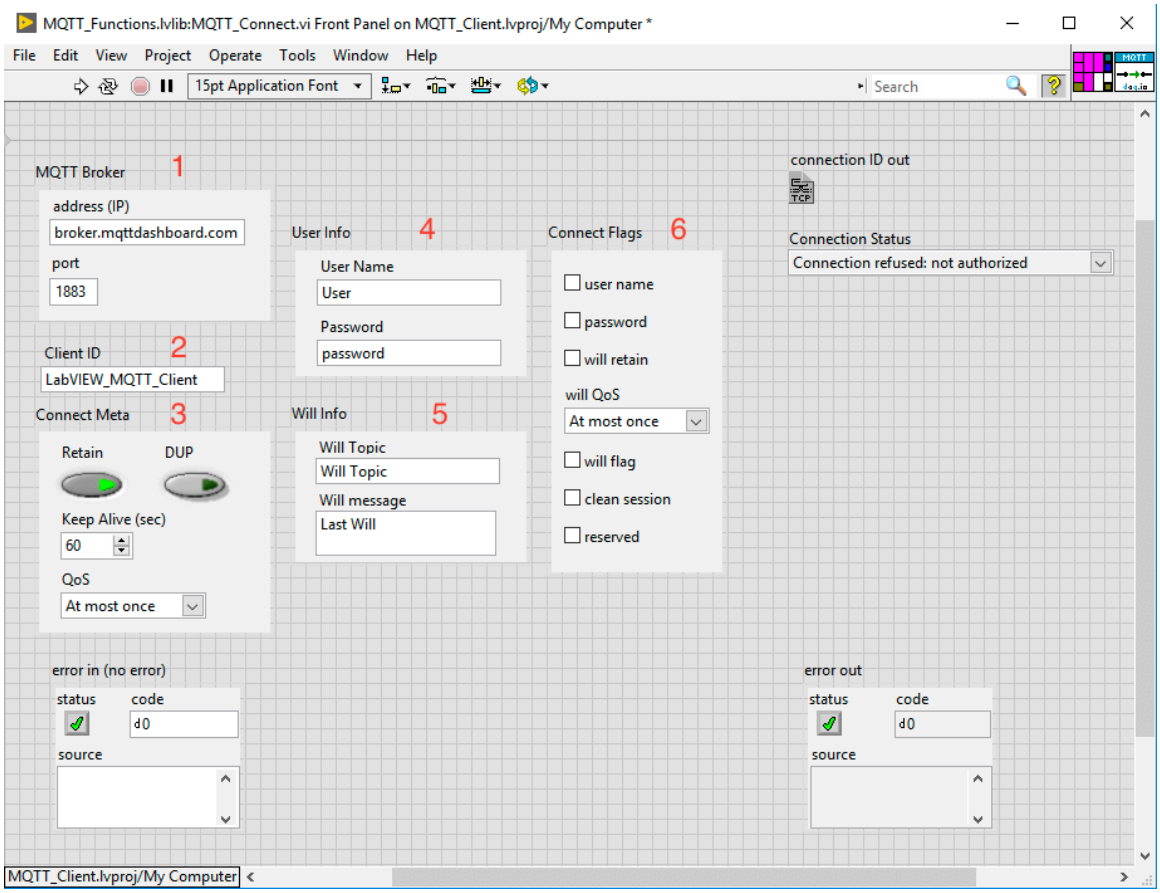


Figura 36: PF MQTT_Connect.vi

Como el DB es muy amplio y se puede mostrar en una misma Figura, se analiza en tres partes, que son las más importantes:

En la parte 1, mostrada en la Figura 37, se observa:

[1] Se trata de abrir una conexión TCP con el *Broker* en la dirección y el puerto especificado en MQTT *Broker*. Si no hay errores se entrega un Identificador de la conexión.

[2] Se establece el estatus de la conexión. Las diferentes opciones se muestran en la Figura 38.

[3] Si no se consigue establecer la conexión TCP y aparece algún error, el resultado a la salida de este comparador será 0 y no se puede establecer comunicación con el *Broker*.

Si, por el contrario, se consigue establecer la conexión TCP, se establece el estatus de conexión “Conecction Accepted” y la salida del comparador será 1 propiciando la comunicación entre *Broker* y cliente, pasando a la parte 2 del DB.

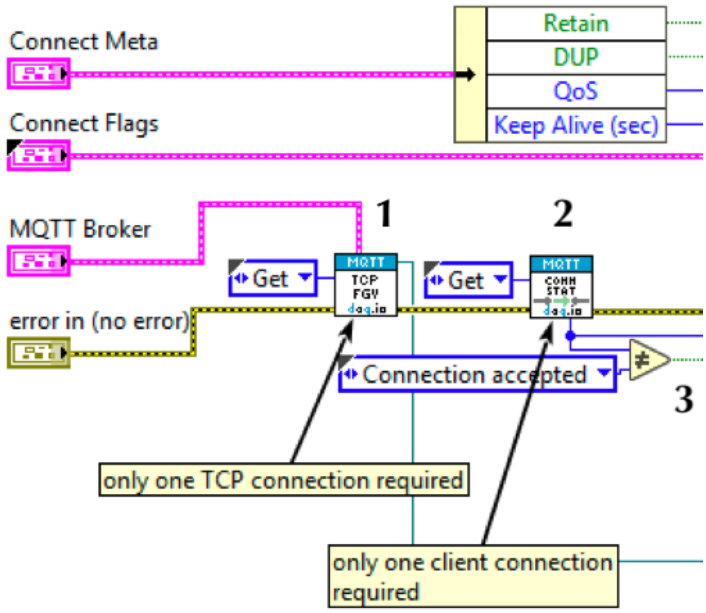


Figura 37: DB MQTT_Connect.vi (parte 1)

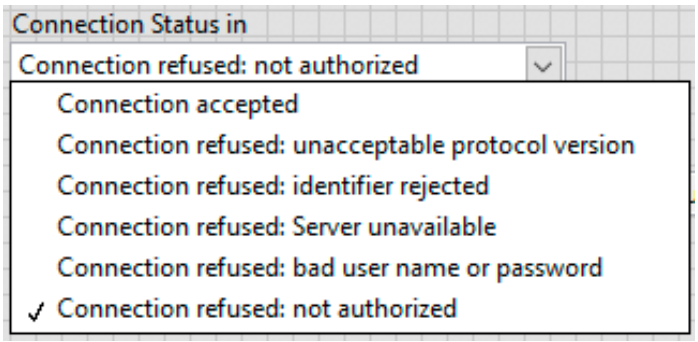


Figura 38: Connection Status

En la Figura 39 se muestra la parte 2 del DB, que a su vez está divida en varias partes:

[1] Se construye la cabecera del mensaje CONNECT a partir de los datos suministrados en Connect Flag y el Keep Alive.

[2] Se construye parte del contenido (payload) del mensaje. Se comienza con el Client ID.

[3] Si el Flag de Will está activo, se añade al contenido del mensaje el Will *Topic* y el Will Message.

[4] Si el Flag de User Name está activo, se añade al contenido del mensaje el User Name.

[5] Si el flag de password está activo, se añade al contenido del mensaje el password.

[6] Se construye el mensaje CONNECT a partir de la cabecera obtenida en [1] y del contenido del mensaje obtenido a partir de [2], [3], [4] y [5].

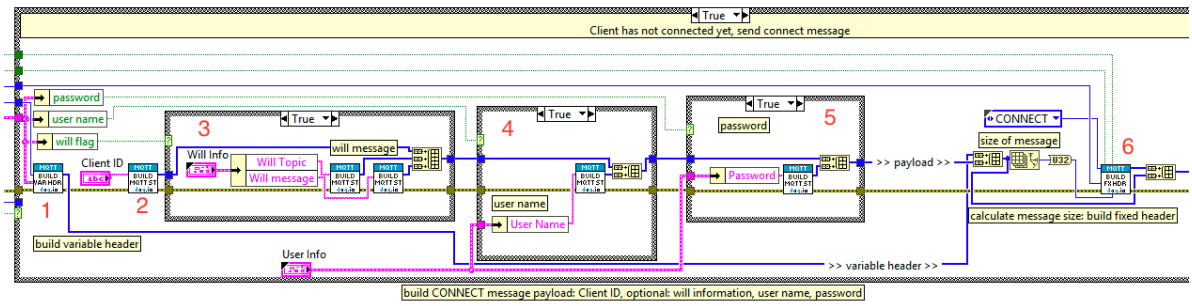


Figura 39: MQTT_Connect.vi (parte 2)

En la Figura 40 se muestra la parte 3 y ultima del DB:

[1] Con el Identificador de conexión generado en la parte 1 se envía por TCP el mensaje CONNECT.

[2] Se lee en mensaje CONNACK que debe ser enviado por el *Broker* al recibir el mensaje CONNECT.

[3] Con el recibimiento de CONNACK, se queda establecida la conexión si no ha ocurrido ningún error.

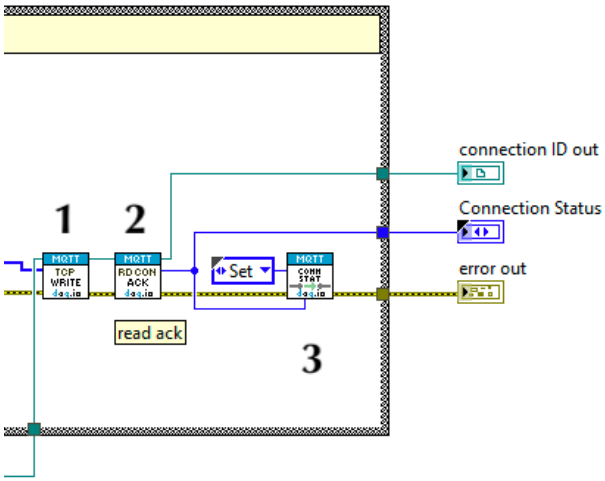


Figura 40: MQTT_Connect.vi (parte 3)

5.2 MQTT_Disconnect.vi

Este VI es el encargado de realizar la correcta desconexión, por parte del Cliente, entre el Cliente y el *Broker*.

En la Figura 41, se muestra el icono y la entrada al VI.

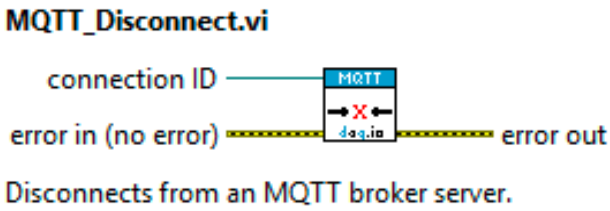


Figura 41: MQTT_Disconnect.vi

El PF de este VI no nos proporciona ninguna información, por lo tanto, no se analiza. En cambio, el DB mostrado en la Figura 42, muestra cómo se realiza la desconexión.

[1] Se construye la cabecera del mensaje DISCONNECT.

[2] Se envía el mensaje al Identificador de conexión.

[3] Se cierra la conexión TCP.

[4] Pone a 0 el Identificador de conexión

[5] Establece el estado de conexión como Servidor no disponible.

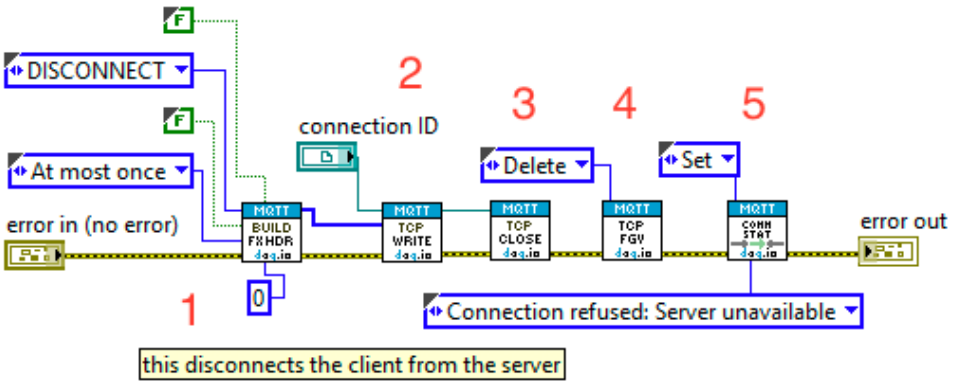


Figura 42: DB MQTT_Disconnect.vi

5.3 MQTT_PingReq.vi

Este VI es el encargado de enviar una petición de ping al *Broker* para mantener viva la conexión entre el Cliente y el *Broker* y que no se realice una desconexión indeseada.

En la Figura 43, se muestra la entrada al VI y las salidas. Estas salidas no son necesarias para su posterior procesamiento.

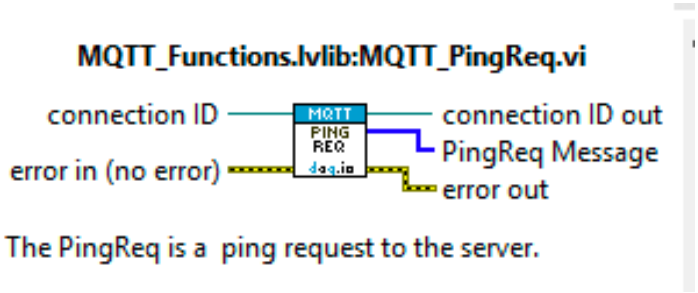


Figura 43: MQTT_PingReq.vi

Como pasa con el anterior VI, el PF no proporciona ninguna información útil. En cambio, en la Figura 44, se observa que en el DB hay dos partes claras:

[1] Se forma la cabecera y el mensaje PingReq que se quiere transmitir al *Broker* junto con el QoS que deseamos, en este caso se emplea QoS 1 para que el *Broker* responda con PINGRESP.

[2] Se envía el mensaje PINGREQ con el identificador de conexión a través de TCP al *Broker*.

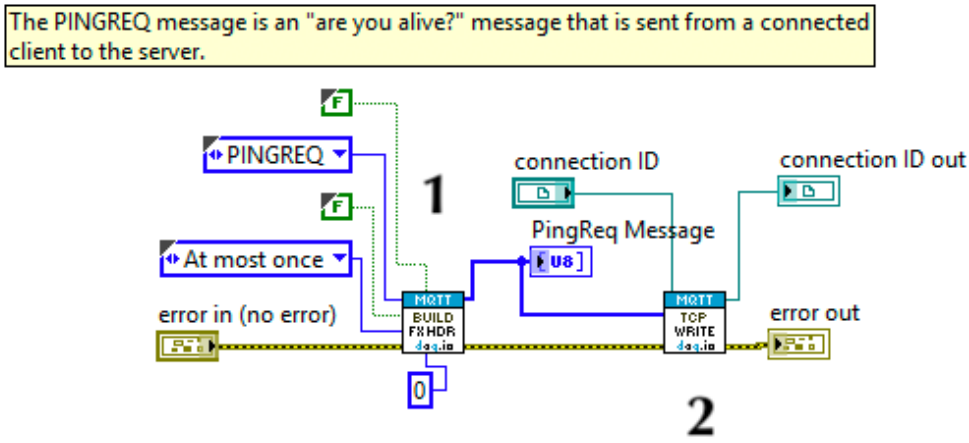


Figura 44: DB MQTT_PingReq.vi

5.4 MQTT_Publish.vi

Este VI es el encargado de enviar el mensaje PUBLISH al *Broker*.

En la Figura 45, se muestra el icono del VI, donde aparecen los parámetros de entrada y la salida del mismo. Esta salida debe ser la misma que la entrada, dado que se trata del Identificador de la conexión.

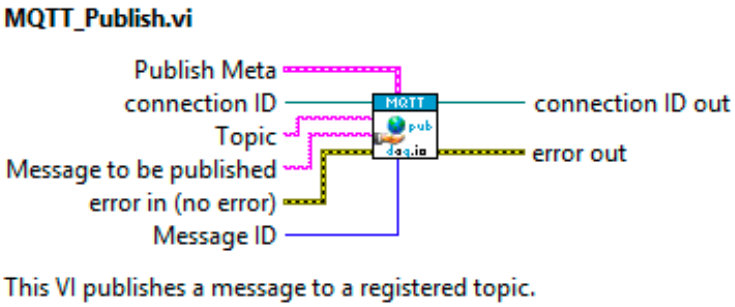


Figura 45: MQTT_Publish.vi

Analizando el VI, se observa en el PF, como se muestra en la Figura 46, que los parámetros necesarios para publicar un mensaje son:

[1] Identificador de conexión: único para cada sesión.

[2] Publish Meta: establece los parámetros de la conexión: QoS, indicación de mensajes duplicados y de mensajes retenidos.

[3] *Topic* del mensaje

[4] Contenido del mensaje a publicar

[5] Identificador de mensaje: con este identificador único para cada mensaje, se puede saber si el mensaje ha llegado cuando el QoS es 1 ó 2. Una vez

asegurada la recepción del mensaje, se puede volver a reutilizar el identificador de mensaje.

Analizando el DB, mostrado en la Figura 47, se observa que:

[1] Se construye la cabecera del mensaje PUBLISH indicando el QoS, el identificador del mensaje y el *Topic* del mensaje.

[2] Transforma el texto del mensaje a codificación UTF-8.

[3] Construye el mensaje uniendo la cabecera generada en [1], el mensaje de [2] y gestionando si se trata de un mensaje duplicado o retenido.

[4] Se envía el mensaje junto con el identificador de conexión a través de TCP al *Broker*.

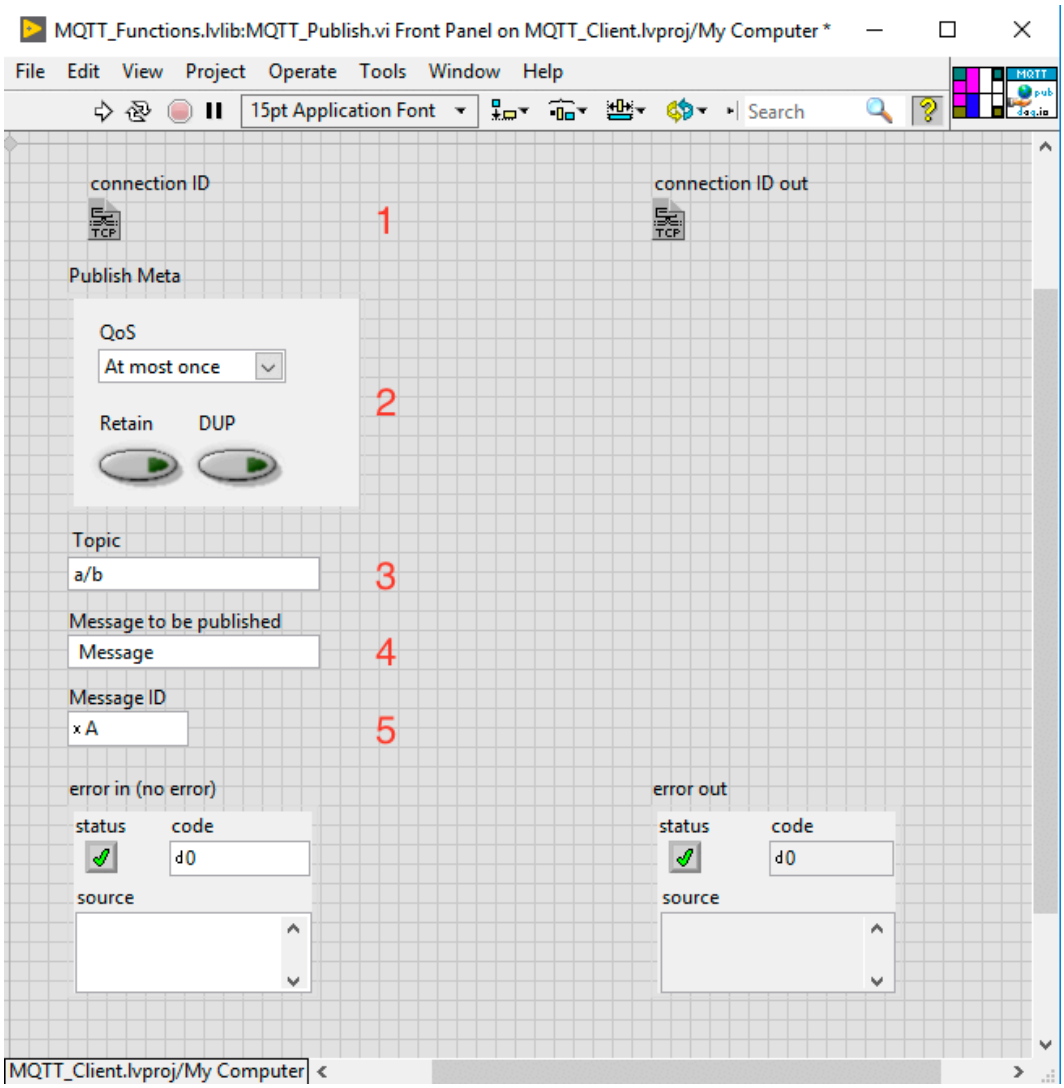


Figura 46: PF MQTT_Publish.vi

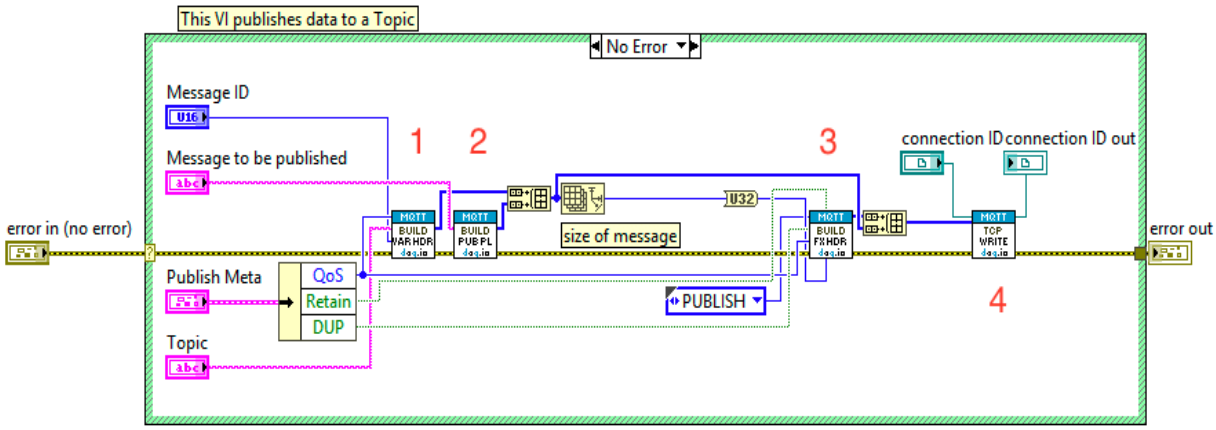


Figura 47: DB MQTT_Publish.vi

5.5 MQTT_Read_Published_Message.vi

Este VI es el encargado de leer un mensaje enviado por el *Broker*.

En la Figura 48, se muestra su icono, el identificador de conexión como entrada y las salidas del mismo. Cabe descatar que las salidas del VI son el Identificador del mensaje, el tipo, el *Topic* y el contenido del mensaje. Estas salidas se pueden utilizar para un posterior procesamiento del mensaje.

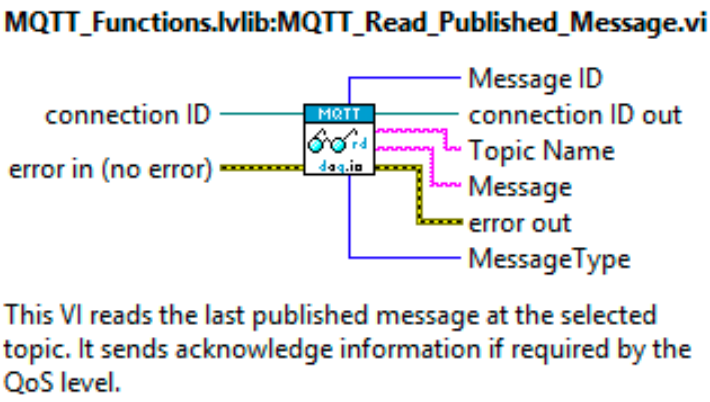


Figura 48: MQTT_Read_Published_Message.vi

Analizando el PF, mostrado en la Figura 49, se pueden apreciar cuatro partes importantes. En este caso, estos parámetros son las salidas del VI anteriormente mencionadas.

[1] Tipo de mensaje: indica de qué tipo de mensaje se trata, si es: PUBLISH, PUBREC, PUBREL, RESERVED, CONNACK, SUBACK, UNSUBACK.

[2] Identificador de mensaje. Como se ha mencionado anteriormente, único e intransferible para cada mensaje, hasta que es recibido.

[3] *Topic* del mensaje.

[4] Contenido del mensaje.

Analizando el DB que se muestra en la Figura 50, se observa que:

- Lee la cabecera del mensaje recibido para averiguar el QoS y el tipo de mensaje publicado.
- Se observa el tipo de mensaje:
  i) Si el tipo de mensaje no es PUBLISH aparece un mensaje de error, mostrando a la salida el tipo de mensaje.
  ii) Si es de tipo PUBLISH se sigue con los siguientes pasos.
- Se realiza una lectura TCP.
- Se lee la cabecera del mensaje para obtener el *Topic* y dependiendo del QoS se elimina el identificador de mensaje o se mantiene. El resto del mensaje se pasa a [5].
- Se pasa el texto de codificado en UTF8 a String para poder leer el contenido del mismo.
- Dependiendo del QoS, se ejecuta un caso u otro.
  1) En el caso que se muestra en la Figura 50, se propaga el identificador de conexión a la salida, dado que el QoS es 0 y no se precisa de ACK.
  2) En el caso que se muestra en la Figura 51, el QoS es 1, y por lo tanto se tiene que enviar el mensaje PUBACK.
  3) En el caso que se muestra en la Figura 52, el QoS es 2, por lo tanto, hay que enviar el mensaje PUBREC, leer el PUBREL y enviar el PUBCOMP.

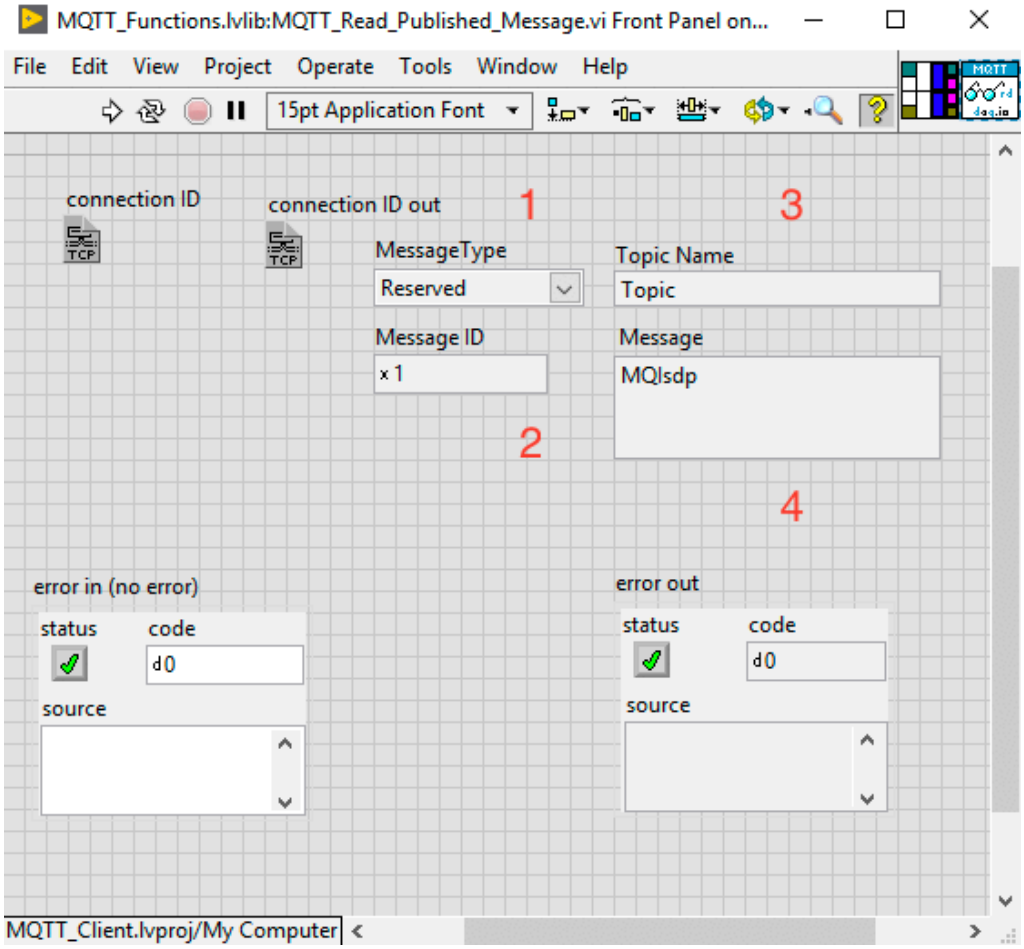


Figura 49: PF MQTT_Read_Published_Message.vi

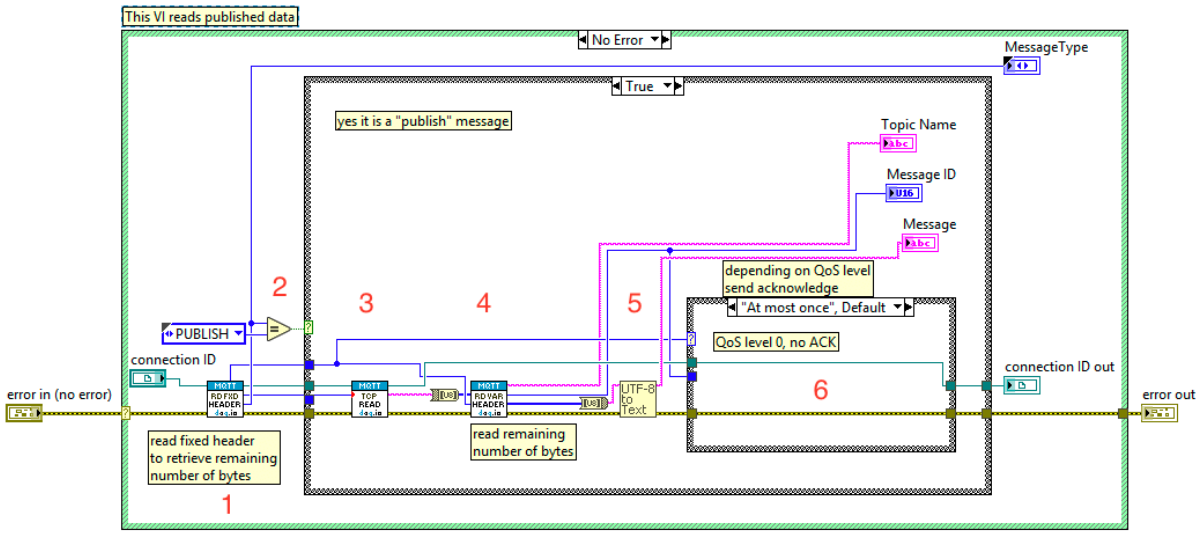


Figura 50: DB MQTT_Read_Published_Message.vi

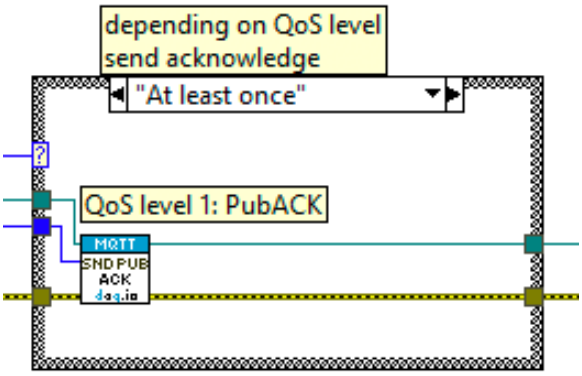


Figura 51: Read_Published QoS1

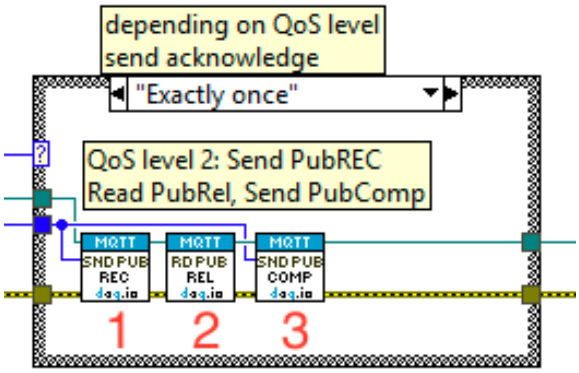


Figura 52: Read_Published QoS2

5.6 MQTT_Subscribe.vi

Este VI es el encargado de suscribirse a uno o varios *Topics* en el *Broker* por parte del Cliente, para poder comenzar a recibir los mensajes publicados en él.

En la Figura 53, se muestra el icono del VI y los parámetros de entrada y la salida del mismo. Si no ha ocurrido ningún error, el connection ID de la entrada se propaga a la salida.

Analizando el PF que se muestra en la Figura 54, se observa que se introducen dos parámetros importantes a la entrada:

[1] Se introduce un identificador de mensaje.

[2] Se escribe el grupo de *Topics* y la calidad de servicio asociada con cada uno de los *Topics*. Se pueden introducir tantos *Topics* como se desee.

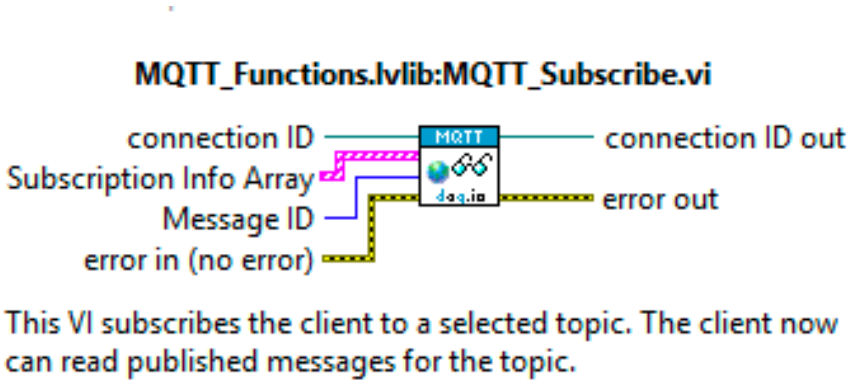


Figura 53: MQTT_Subscribe.vi

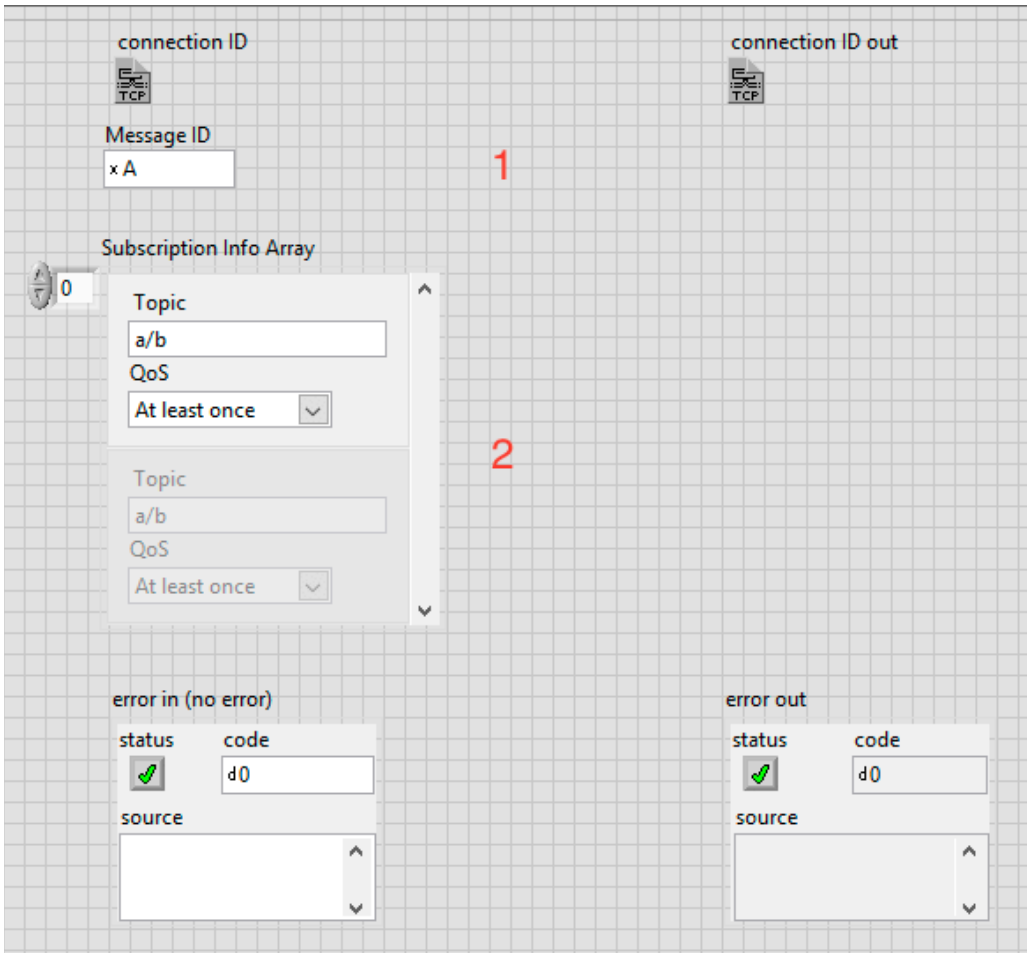


Figura 54: PF MQTT_Subscribe.vi

Analizando el DB que se muestra en la Figura 55, se observan tres partes diferenciadas:

[1] En el bucle For, se crea el contenido del mensaje, compuesto por un array con el conjunto: Identificador de mensaje, *Topic* ya en codificación UTF8 y el QoS asociado a cada uno.

[2] Se construye la cabecera del mensaje SUBSCRIBE.

[3] Se envía a través de TCP el mensaje SUBSCRIBE formado por la cabecera obtenida en [2] y el contenido del mensaje obtenido en [1], con el identificador de conexión que se introduce como parámetro a la entrada.

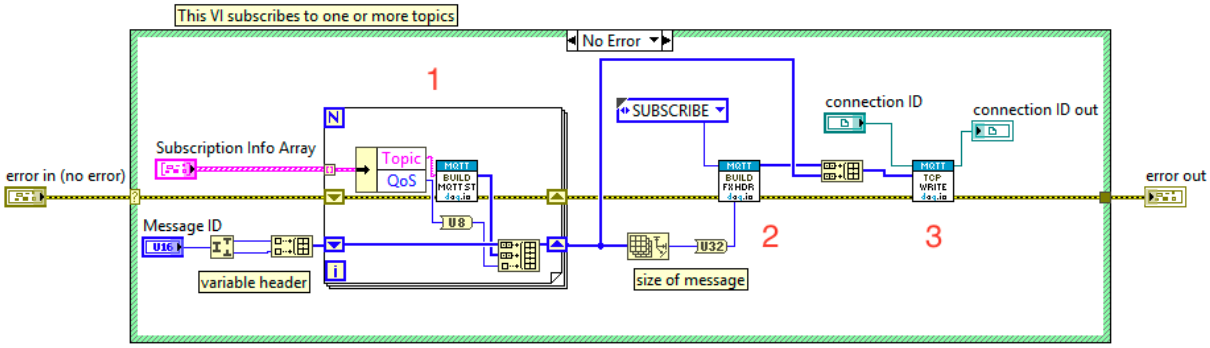


Figura 55: DB MQTT_Subscribe.vi

5.7 MQTT_Unsubscribe.vi

Este VI es el encargado de cancelar la suscripción de uno o varios *Topics* por parte del Cliente.

En la Figura 56, se muestra el icono del VI y las entradas y salidas del mismo. Si no ha ocurrido ningún error, el connection ID de la entrada se propaga a la salida.

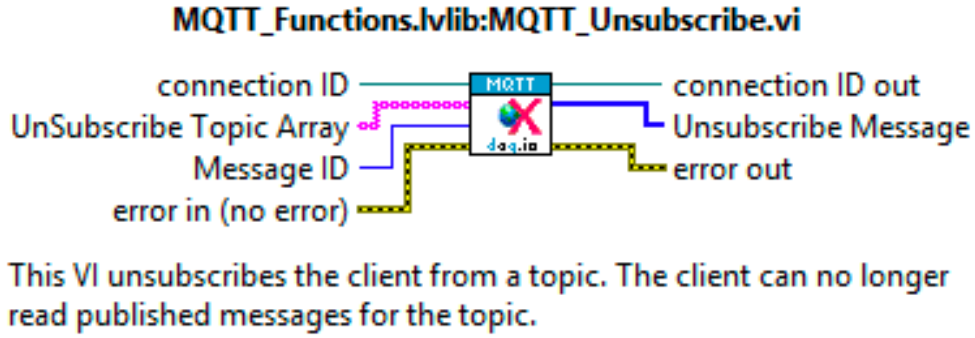


Figura 56: MQTT_Unsubscribe.vi

Analizando el PF que se muestra en la Figura 57, se observa que los parámetros importantes en este VI son:

[1] Identificador de mensaje.

[2] Array de *Topics* a cancelar suscripción.

[3] Lista de mensajes UNSUBSCRIBE a enviar al *Broker*.

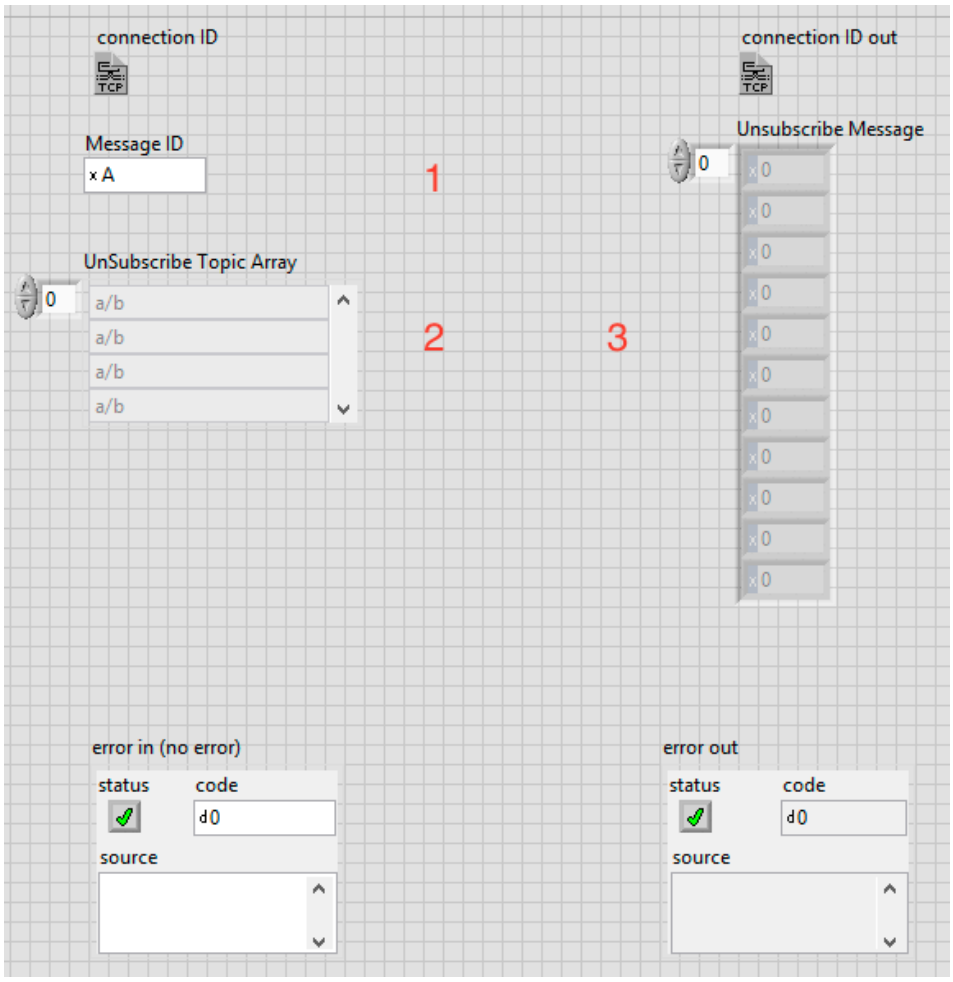


Figura 57: PF MQTT_Unsubscribe.vi

Analizando el DB que se muestra en la Figura 58, se observa como en el VI anterior que existen tres partes:

[1] En este bucle For, se crea el contenido del mensaje, compuesto por un array con el conjunto: Identificador de mensaje y *Topics* a cancelar suscripción codificado en UTF8.

[2] Se construye la cabecera del mensaje UNSUBSCRIBE con un QoS 0.

[3] Se envía a través de TCP el mensaje UNSUBSCRIBE formado por la cabecera obtenida en [2] y el contenido del mensaje obtenido en [1], con el identificador de conexión que se introduce como parámetro a la entrada.

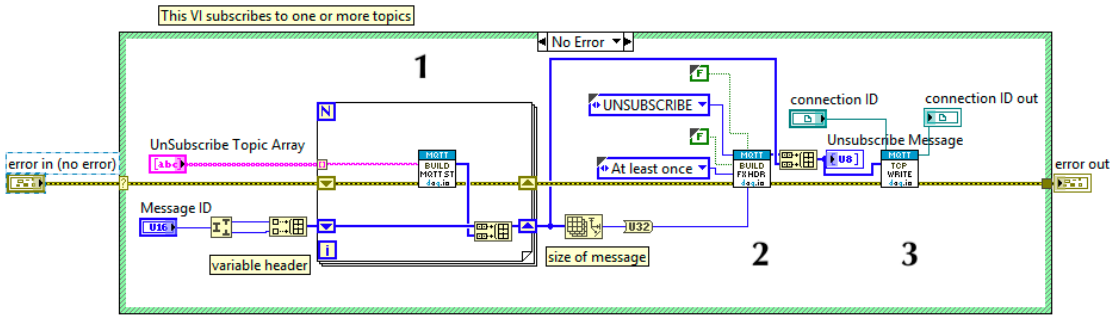


Figura 58: DB MQTT_Unsubscribe.vi

6 Caso práctico: Simulación de automatización de procesos en un hotel

Como caso práctico para comprobar las utilidades y ventajas que tiene el empleo del protocolo de comunicaciones MQTT en un sistema IIoT, se pretende realizar un sistema de control para automatizar algunos procesos comunes y otros eventuales que pueden suceder en un hotel.

Los procesos escogidos son la automatización de la luminaria de un hotel en función de unos escenarios determinados y la actuación de otros dispositivos en caso de emergencia de incendio.

Se va a realizar un pequeño escenario de demostración, que puede ser ampliable añadiendo un mayor número de dispositivos, sin modificar el funcionamiento del sistema completo. A su vez se pueden ampliar automatizaciones sin necesidad de modificar todo el sistema de control.

Como protocolo de comunicaciones para la interoperabilidad de los dispositivos y el programa de automatización, se emplea la versión 3.1.1 de MQTT. Esta versión está sobradamente probada y en pleno funcionamiento. Si se deseara portar el control de la automatización a la versión 5.0 del protocolo, habría que hacer una adecuación de todo el sistema dado que existen grandes diferencias entre ambos protocolos, tal y como se ha comentado en el capítulo 4.3.

Para el desarrollo y depuración del sistema de automatización, se emplea la herramienta de programación de NI, *Labview*, que se ha estudiado en este TFG, utilizando como base para la programación las librerías que se han explicado en el capítulo 5 y realizando algunas modificaciones sobre algunas de ellas.

6.1 Especificaciones del sistema

Las especificaciones básicas del sistema son:

1) Luminaria:
   a) Existencia de luz exterior:
      i) Pasillos o zonas que tienen acceso de luz desde el exterior, no encender las luces mientras haya suficiente luz externa.
      ii) Pasillos o zonas que no tienen acceso de luz desde el exterior, encender o apagar según presencia.
   b) No existe luz exterior:
      i) Todos los pasillos y zonas, se debe encender o apagar la luz según presencia.
2) Sistema de incendio:
   a) No hay detección de humo:
      i) Proceder de manera normal.

b) Hay detección de humo
   i) Actuar sobre las puertas antincendios, permitiendo su cierre.
   ii) Encender todas las luces y luces de emergencia.
   iii) Hacer sonar la alarma antincendios.

3) Interfaz de usuario que permita ver en tiempo real qué sucede en los sensores.

Cuando se detecte la presencia de una persona: se debe encender la luz de la zona donde se encuentra y las zonas anterior y posterior adyacentes, de manera que la luz vaya adelantando a la persona pudiendo facilitarle la visión a lo largo del pasillo.

6.2 Diseño del sistema completo

Para la realización del sistema de automatización, y cumplir con las especificaciones anteriormente mencionadas, se debe dividir el sistema en varias partes o bloques:

1. Despliegue de *Broker* MQTT

   Se trata de instalar en un PC un *Broker mosquitto* que haga de interfaz entre los diferentes clientes.

2. Diseño de simuladores

   Dentro de este bloque hay dos partes diferenciadas:

   [1] Diseño de simulación de dispositivos

   Con estos proyectos, se pretende simular el comportamiento de los dispositivos involucrados en la automatización de procesos del hotel.

   Para ello, se pueden agrupar los dispositivos a simular en dos grandes grupos:

   - Sensores:

     Son dispositivos que actúan como clientes MQTT que publican mensajes hacia el *Broker* con un *Topic* especifico. En este grupo se encuentran los detectores de presencia, de luz y de humo.

   - Actuadores:

     Son dispositivos que actúan como clientes MQTT suscritos a los *Topics* creados por los sensores. En este grupo se encuentran las bombillas, las luces de emergencia, la alarma y los relés de las puertas.

[2] Diseño de simulación del interfaz de usuario

Se trata de un proyecto en el que se simula en tiempo real los diferentes escenarios contemplados en las especificaciones. Dentro del proyecto se encuentra la librería de variables globales, necesarias para poder vincular los componentes de este interfaz con cada dispositivo simulado.

3. Diseño del programa de control de la automatización.

   Se trata de un proyecto que contiene el control de la automatización del sistema. Esta aplicación, se conecta con el *Broker* y se encarga de gestionar el control necesario para cumplir la especificación del sistema, mandando mensajes de actuación al *Broker* para que los diferentes dispositivos suscritos a los *Topics* realicen su función.

6.3 Despliegue *Broker*

Lo primero que se tiene que realizar es la instalación del *Broker*. Dentro de la gran variedad de *Brokers* que existen en la red, se decide instalar *mosquitto* porque es un *Broker* ligero, sencillo y que soporta la versión 3.1.1 del protocolo MQTT.

Para ello, descargamos el ejecutable para Windows desde la página de *mosquitto* [50]. Ejecutamos el archivo y seleccionamos la ubicación donde queremos que se instale.

Una vez instalado en el PC, para lanzar el proceso, abrimos una ventana de comandos y en la ubicación donde está instalado el *Broker*, lanzamos el comando “mosquitto –v”.

Este comando nos permite arrancar el *Broker* con la configuración por defecto: puerto de escucha 1883, y nos muestra en pantalla todo el tráfico que pasa por el *Broker*.

Para comprobar que se ha desplegado el *Broker* y que funciona correctamente, se emplea la herramienta de simulación de clientes MQTTBox.

Con ella se realizan suscripciones de ejemplo y publicaciones de prueba para comprobar el funcionamiento. En la Figura 59 se muestra un ejemplo de cliente suscrito al *Topic* “PROYECTO/PRUEBA” y otro cliente que escribe en ese mismo *Topic*.

En la Figura 60 se observa un pantallazo de la ventana de comandos con el *Broker* desplegado, en el que se puede apreciar la recepción del mensaje por parte del cliente que publica y el envío del mensaje al cliente suscrito al *Topic*.

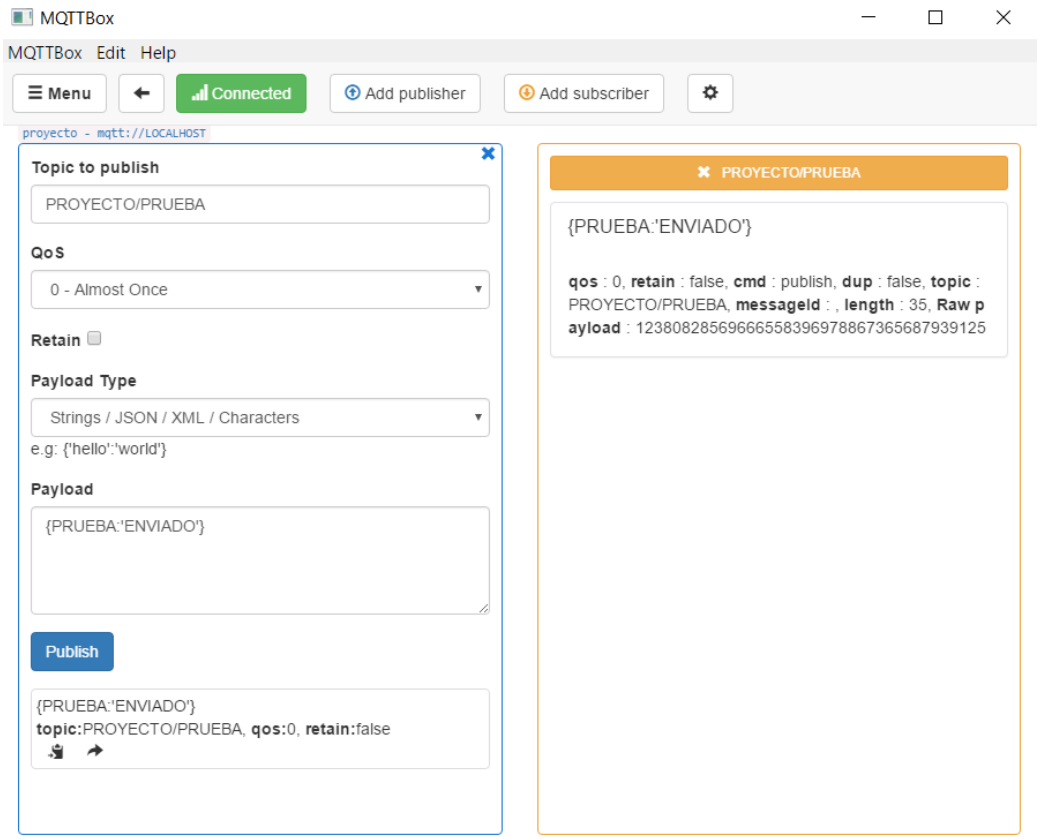


Figura 59: Ejemplo clientes MQTTBox

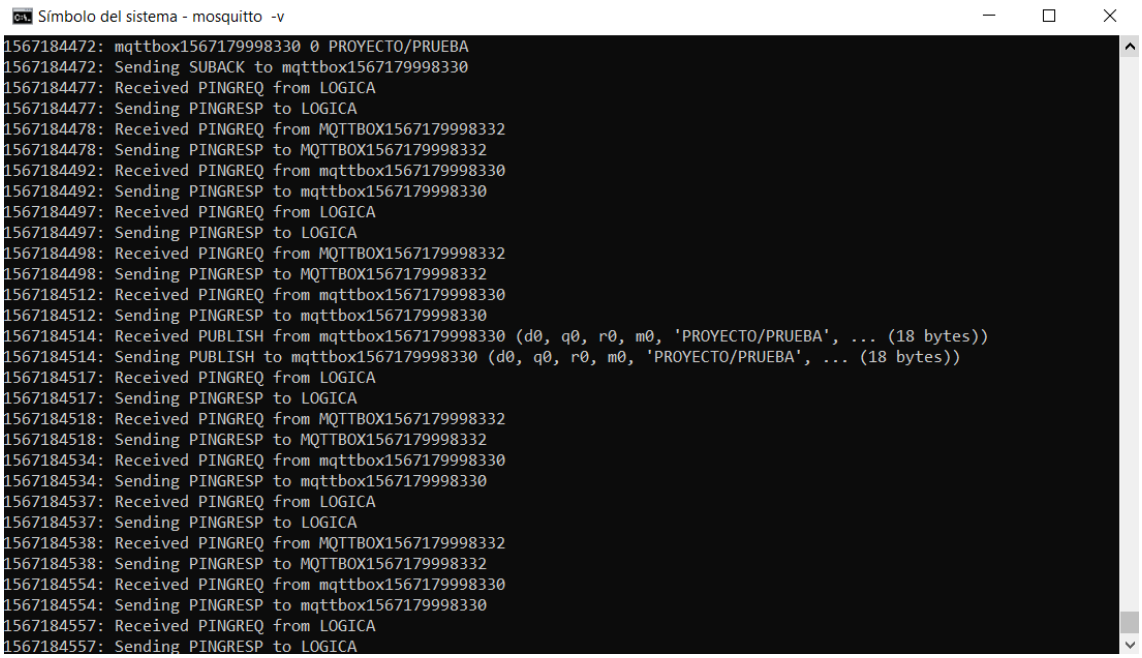


Figura 60: Muestra de funcionamiento Broker mosquitto

6.4 Diseño de Simuladores

Como se ha mencionado en el capítulo 6.2, este bloque se divide a su vez en dos grandes bloques que realizan la simulación del sistema. Se decide realizar esta simulación para comprobar si el control realizado es el correcto antes de realizar el montaje final.

Con estos proyectos que contienen las simulaciones, se podrían realizar todo tipo de simulaciones para otras especificaciones, dado que cualquier dispositivo se puede comportar como un sensor, publicando un mensaje, o como un actuador, suscribiéndose a un *Topic*.

A continuación, se va a profundizar en estos dos tipos de proyectos, explicando más detalladamente qué y por qué se ha realizado de esta manera.

6.4.1 Diseño de simuladores de dispositivos

Como cada dispositivo se va a convertir en un Cliente MQTT, cada dispositivo debe ir implementado en un proyecto, de manera que cuando se quiera montar el sistema final, solo haya que sustituir el proyecto de simulación del dispositivo, por el dispositivo real.

Atendiendo a los dos grupos de dispositivos, se detalla un proyecto de cada grupo de dispositivos, haciendo extensible la explicación al resto de dispositivos del grupo.

Estos proyectos están formados por dos VI:

[1] VI de llamada

Se trata de un grupo de VIs de tránsito empleados para introducir a la variable global en el dispositivo y ejecutar el mismo.

[2] Vi principal

Son los VI que contienen la funcionalidad propia del dispositivo.

Para poder integrar estos simuladores de dispositivos en el resto del sistema, se debe tener una librería con las variables globales que se van a emplear en los diferentes proyectos de simuladores de dispositivos. Esta librería se explicará en el diseño de la interfaz de usuario.

VI de llamada

Hay dos grupos de VIs de llamada, dependientes del tipo de dispositivo que se desea simular. En el caso de estudio, hay un total de 9 VIs RUNPx y 9 VIs RUNx.

[1] RUNP – Para sensores

En la Figura 61, se muestra el icono y la descripción de este grupo de VIs.

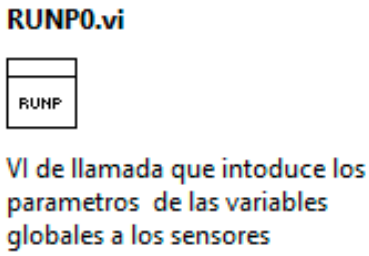


Figura 61: Icono y descripción RUNPx.vi

Analizando el DB mostrado en la Figura 62, se puede observar que se inicia una conexión para trabajar con una variable global y se establece el número de serie del dispositivo.

En este caso, los parámetros de la variable global son:

- Permisos de acceso de lectura requerido y ninguno de escritura.
- Se va a trabajar con Booleanos
- Se especifica el nombre y la ubicación de la variable global.

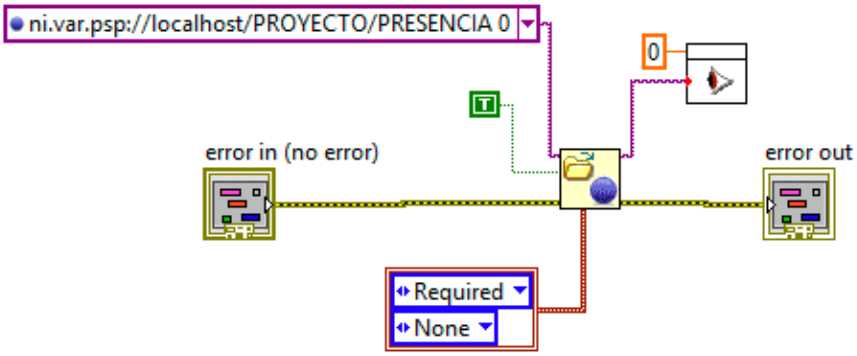


Figura 62: DB RUNPx.vi

[2] RUN – Para actuadores

En la

Figura 63, se muestra el icono y la descripción de este grupo de VIs.

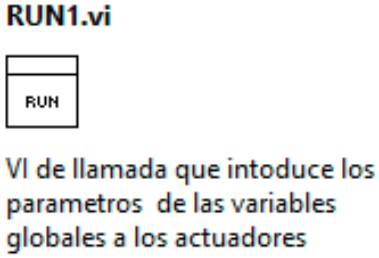


Figura 63: Icono y descripción RUNx.vi

Analizando el DB mostrado en la Figura 64, se puede observar que se inicia una conexión para trabajar con una variable global y se establece el número de serie del dispositivo.

En este caso, los parámetros de la variable global son:

- Permisos de acceso de escritura requerido y ninguno de lectura.
- Se va a trabajar con Booleanos
- Se especifica el nombre y la ubicación de la variable global.

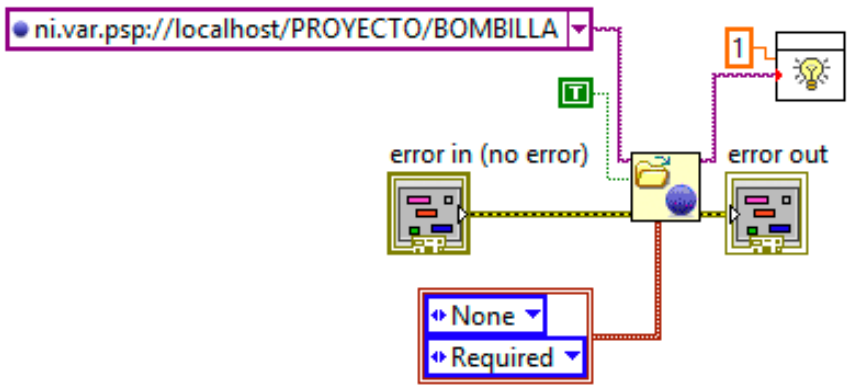


Figura 64: DB RUNx.vi

VI principal

Dentro de este grupo de VIs, tenemos dos grupos:

[1] Sensores

Los sensores son dispositivos que envían mensajes PUBLISH en función de su estado.

En este caso de estudio, se simulan:

- Sensor de luz
- Sensor de presencia
- Sensor de humo

La diferencia entre ellos es el nombre y el número de serie del dispositivo, y el *Topic* y el contenido del mensaje.

En la Figura 65 se muestra el PF de un dispositivo Sensor. Este PF es común para todos los dispositivos de este grupo.

Se observa que hay:

i) Un indicador de conexión establecida, que se activará mientras la sesión MQTT este activa.

ii) Un indicador donde aparecerá el nombre con el que se le conocerá al dispositivo en la conexión.

iii) Un interruptor que iniciará la publicación del mensaje.

iv) Un interruptor que solicite el fin de la conexión.

Figura 65: PF dispositivo Sensor

Si observamos el DB de Sensor_Luz.vi, podemos diferenciar 5 partes o bloques.

En la Figura 66 se observa la primera parte del DB. En este primer bloque, se realiza la conexión del dispositivo con el *Broker* MQTT y se introducen los parámetros necesarios para ella.

En este caso, se modifica la librería, con el fin de que el bloque MQTT_Connect.vi no tenga que esperar el mensaje CONNACK, gestionando ese mensaje en otro bloque del DB.

Los parámetros introducidos para la conexión de los dispositivos son muy básicos. Se establece:

i) No hay sesión persistente, ni mensajes duplicados ni retenidos.

ii) Se especifica que el QoS sea 0

iii) El Keep Alive se configura en 20 segundos

iv) La dirección IP y el puerto donde se encuentra el *Broker* son 192.168.1.51:1883.

v) El ClientID dependerá del sensor que estemos simulando. En este caso: sensor_luz_%d, donde %d es el número de serie del dispositivo.

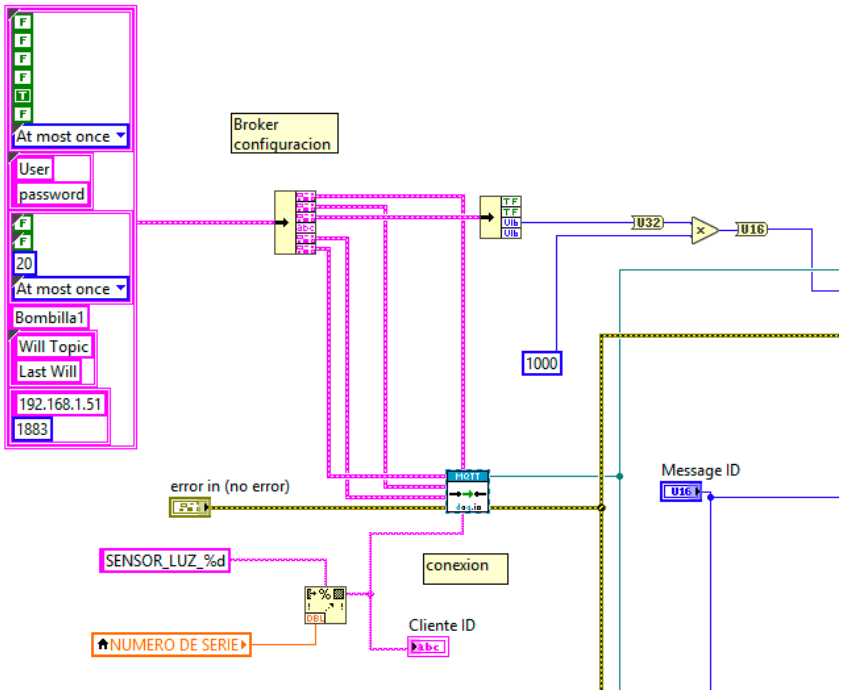


Figura 66: Parte 1 DB Sensor_Luz.vi

En la Figura 67 se observa la segunda parte del DB, bucle de gestión de mensajes. En este bloque, se realiza la gestión de mensajes publicados.

Se gestionan los mensajes CONNACK, PUBACK, PINGRESP y Reserved.

Lo primero que se hace al entrar en el bloque es identificar cuál es el mensaje que hay que gestionar, leyendo la cabecera del mismo. Una vez identificado el mensaje se procede con él:

1) CONNACK

   Se lee el mensaje y activa el indicador de conexión activa.

2) PINGRESP

   Se propaga la entrada a la salida.

3) PUBACK

   Gestiona el mensaje en función del QoS. En este caso, se establece un QoS 1 en la publicación, por lo tanto, hay que atender a la lectura de PUBACK. Están implementadas todas las opciones de QoS por si en un futuro se pretende variar el QoS en el dispositivo.

4) Reserved

   Este caso es para cuando pasa el tiempo establecido en el Keep Alive. En este caso, a la salida de la lectura de la cabecera, aparece

el error 56. Se trata de un error de timeout, por lo tanto, es el que nos indica que no se ha producido intercambio de mensajes con el *Broker* y por lo tanto hay que mandar un PINREQ. Para ello, borramos el error y mandamos el mensaje PINGREQ.

Por último, se realiza la gestión de errores, haciendo que se pare la ejecución del bucle de gestión de mensajes si se emplea el interruptor de parada, o si existe un error en alguno de los bucles del VI.

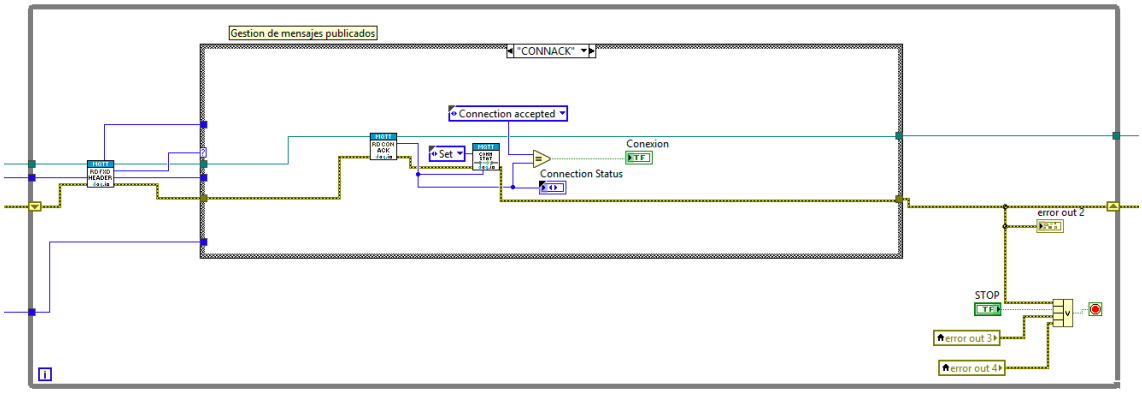


Figura 67: Parte 2 DB Sensor_Luz.vi

En la Figura 68 se observa la tercera parte del DB, bucle de envió de mensaje PUBLISH.

Este bloque, es el encargado en enviar el mensaje PUBLISH cuando se activa el interruptor de publicación. El *Topic* y el contenido del mensaje son siempre los mismos en cada dispositivo. En este caso, el contenido del mensaje enviando es {‘ESTADO’: ‘LUZ’, ‘HORA’: ‘%H:%M:%S’} indicando que es necesaria luz y la hora en la que se manda el mensaje.

Mientras el interruptor este activo, se manda un mensaje cada 500 milisegundos, asegurándonos de este modo que no va a estar enviando mensajes continuamente.

Por último, se realiza la gestión de errores, haciendo que se pare la ejecución del bucle de envió de mensajes si se emplea el interruptor de parada, o si existe un error en alguno de los bucles del VI.

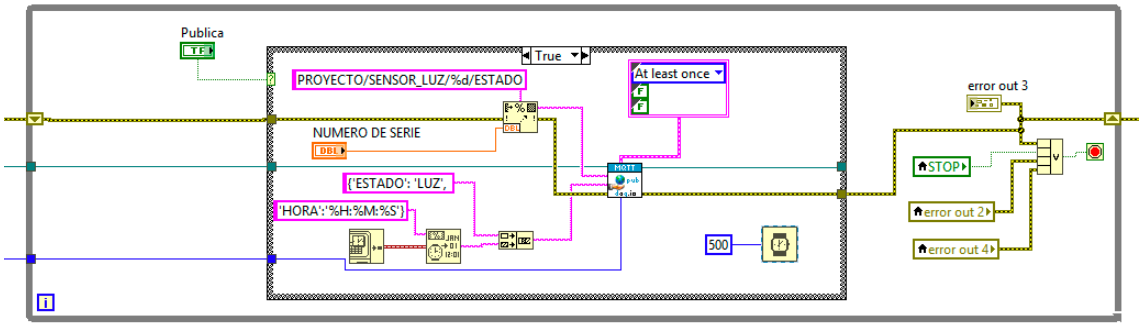


Figura 68: Parte 3 DB Sensor_Luz.vi

En la Figura 69 se observa la cuarta parte del DB, gestión de variable global. Esta es la parte del VI que enlaza el funcionamiento del dispositivo con el interfaz de usuario, haciendo que se lea lo que la variable global asociada tiene y el valor de la misma se pasa al bucle de envío de mensaje para que envíe el mensaje PUBLISH.

Por último, al igual que sucedía en los bloques anteriores, se realiza la gestión de errores, de modo que se pare la ejecución del bucle si hay un error en alguno de los bucles o se pulsa el interruptor de parada.

Si sucede un error en este bucle, se cierra la conexión de la variable global y se propaga el error a la última parte del DB.

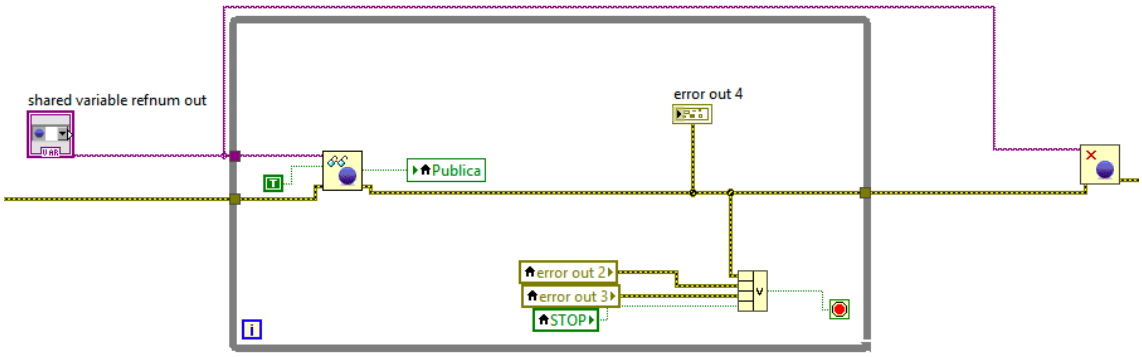


Figura 69: Parte 4 DB Sensor_Luz.vi

Para finalizar en la Figura 70 se observa la última parte del DB, donde se realiza la desconexión del dispositivo con el *Broker* cuando se ha parado la ejecución de los bucles anteriormente comentados. El resultado de esta desconexión se aprecia en el indicador de conexión del PF.

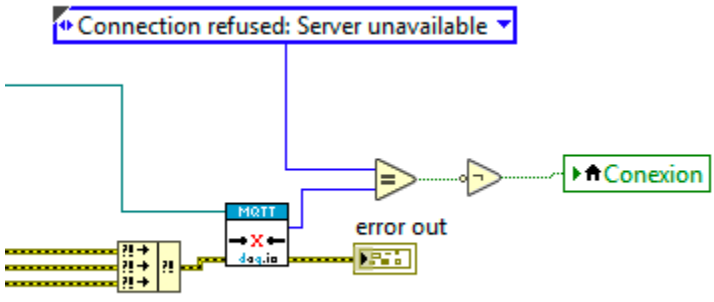


Figura 70: Parte 5 DB Sensor_Luz.vi

[2] Actuadores

Los actuadores son dispositivos que se suscriben a *Topics* y reciben mensajes PUBLISH escribiendo sobre una variable global.

En este caso de estudio, se simulan:

- Bombillas
- Reles de puertas (simulado como si se tratara de una bombilla)
- Alarma de incendios (simulado como una bombilla)

La diferencia entre ellos es el nombre y el número de serie del dispositivo, y el *Topic* al que se suscriben.

En la Figura 71 se muestra el PF de un dispositivo Actuador. Este PF es común para todos los dispositivos de este grupo.

Se observa que hay:

- Un indicador de conexión establecida, que se activará mientras la sesión MQTT este activa.
- Un indicador donde aparecerá el nombre con el que se le conocerá al dispositivo en la conexión.
- Un indicador led que se encenderá simulando a la bombilla.
- Un interruptor que solicite el fin de la conexión.

Figura 71: PF dispositivo Actuador

Si observamos el DB de Bombilla2.vi, vemos 3 partes muy diferenciadas.

En la Figura 72, se observa la primera parte del DB. En ella se realiza la conexión del dispositivo con el *Broker* y tal como sucede con los dispositivos sensores, se fijan los parámetros de configuración de la conexión.

En este caso, se emplea la librería modificada, con el fin de que el bloque MQTT_Connect.vi no tenga que esperar el mensaje CONNACK, gestionando ese mensaje en otro bloque del DB.

Los parámetros introducidos para la conexión de los dispositivos son muy básicos. Se establece:

i) No hay sesión persistente, ni mensajes duplicados ni retenidos.

ii) Se especifica que el QoS sea 0

iii) El Keep Alive se configura en 20 segundos

iv) La dirección IP y el puerto donde se encuentra el *Broker* son 192.168.1.51:1883.

v) El ClientID dependerá del actuador que estemos simulando. En este caso: BOMBILLA_%d, donde %d es el número de serie del dispositivo.

En esta parte del DB, también se observa la inicialización de las variables locales y la entrada de la variable global que se introduce en RUNx.vi asociado al dispositivo y el formato del contenido del mensaje PUBLISH.

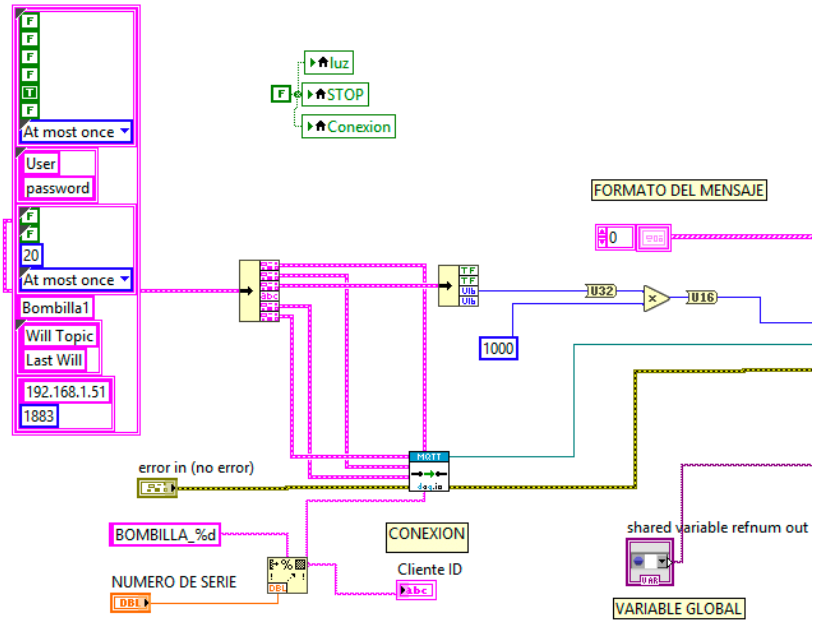


Figura 72: Parte 1 DB Bombilla2.vi

La segunda parte de este DB, mostrado en la Figura 73, se divide a su vez en varias partes, dado que es el bucle de gestión de mensajes. En este bucle, se gestionan mensajes como: CONNACK, PINGRESP, PUBLISH, SUBACK y Reserved.

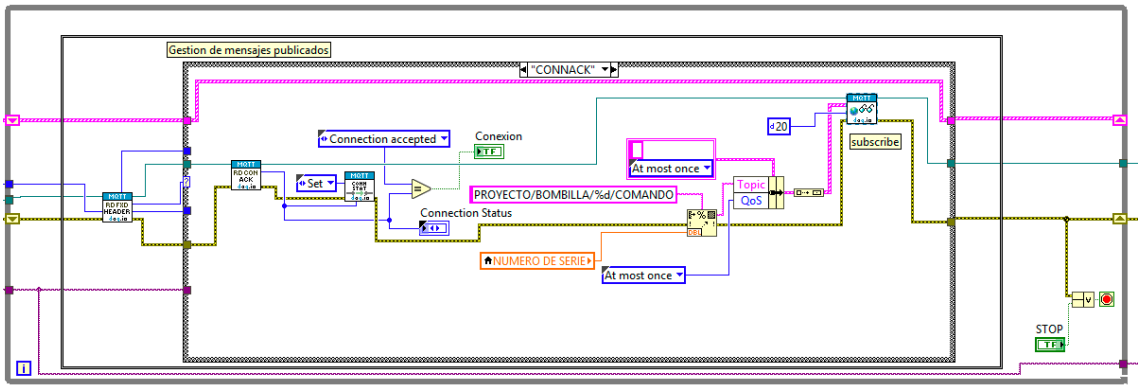


Figura 73: Parte 2 DB Bombilla2.vi

Lo primero que se hace al entrar en el bloque es identificar cuál es el mensaje que hay que gestionar, leyendo la cabecera del mismo. Una vez identificado el mensaje se procede con él:

- CONNACK

  En la Figura 74 se observa la gestión del mensaje CONNACK. En ella se realiza la lectura del mensaje y se activa el indicador de conexión si la conexión se ha establecido con éxito. También se aprovecha y se realiza la suscripción al *Topic* con un QoS 0.

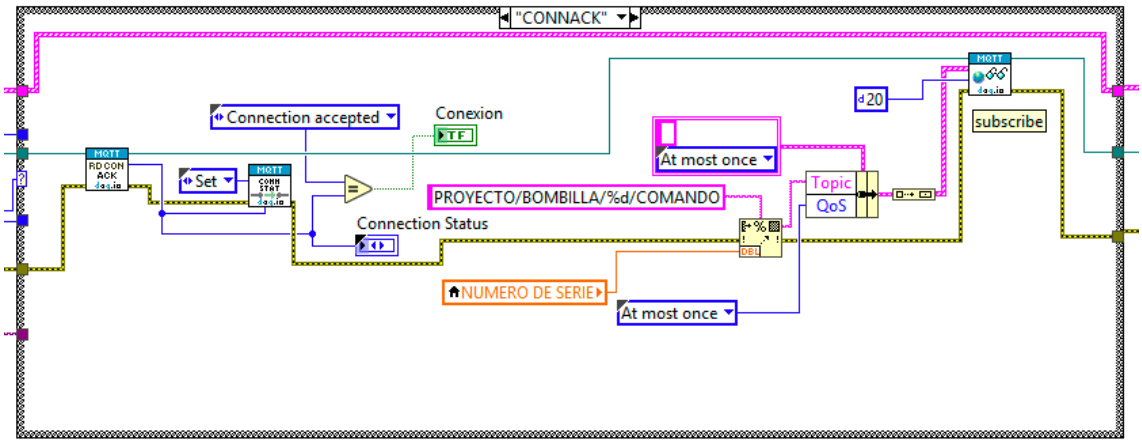


Figura 74: Gestión CONNACK Bombilla2.vi

- PINGRESP

  Se propaga la entrada a la salida.

- SUBACK

  Lee el mensaje. Está implementado para casos en los que la suscripción se realiza con QoS superior a 0. En el caso de estudio, como la suscripción se realiza con QoS 0, no sería necesario el caso.

- Reserved

  Este caso es para cuando pasa el tiempo establecido en el Keep Alive. En este caso, a la salida de la lectura de la cabecera, aparece el error 56. Se trata de un error de timeout, por lo tanto, es el que nos indica que no se ha producido intercambio de mensajes con el *Broker* y por lo tanto hay que mandar un PINREQ. Para ello, borramos el error y mandamos el mensaje PINGREQ. En la Figura 75 se muestra la gestión de este mensaje.

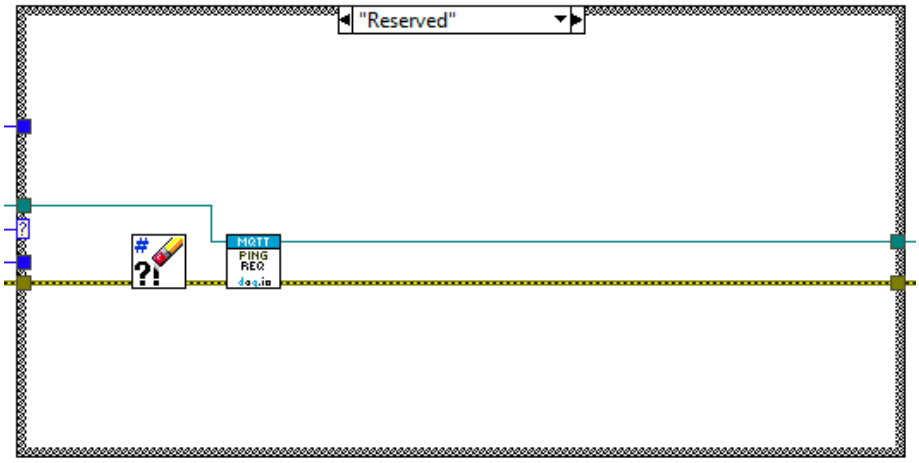


Figura 75: Gestión Reserved Bombilla2.vi

- PUBLISH

  En la Figura 76 se observa la gestión del mensaje PUBLISH. Se lee el mensaje.

  Si el contenido del mensaje es “1” se enciente la luz del PF y se introduce el valor en la variable global asignada en el RUNx.vi asociado.

  Si el contenido del mensaje es “0” se apaga la luz del PF y se introduce el valor en la variable global asignada en el RUNx.vi asociado.

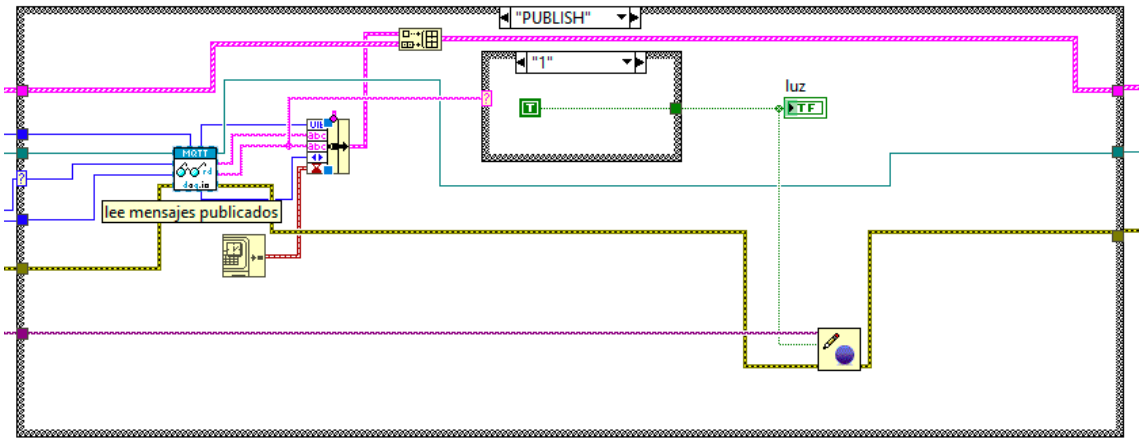


Figura 76: Gestión PUBLISH Bombilla2.vi

Cuando se termina de gestionar cada mensaje, se evalúa si se repite el bucle o ha existido algún error o ha cambiado el estado del interruptor de parada para salir del bucle.

En apartados sucesivos se explicará cual es la jerarquía del sistema y qué actuadores se suscriben a determinados *Topics* y en qué *Topics* publican qué sensores.

Para finalizar, en la Figura 77 se observa la última parte del DB, donde se realiza la desconexión del dispositivo con el *Broker* cuando se ha parado la ejecución del bucle anteriormente comentado y se cierra la conexión de la variable global asociada. El resultado de esta desconexión se aprecia en el indicador de conexión del PF.

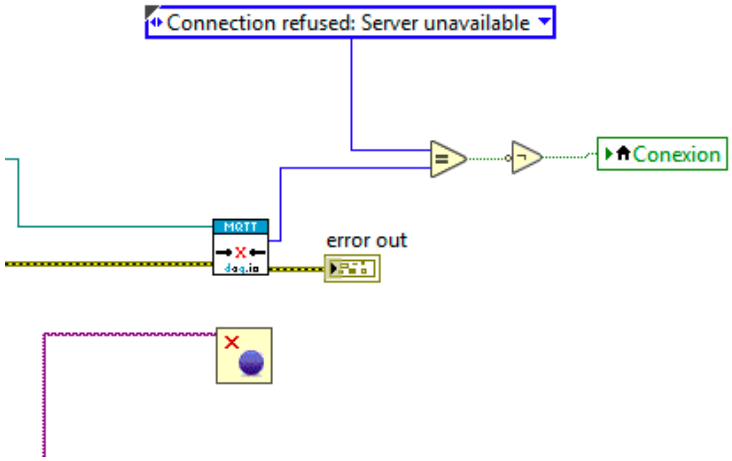


Figura 77: Parte 3 DB Bombilla2.vi

En las siguientes Figuras, se muestran los iconos y descripciones de los diferentes dispositivos actuadores y sensores simulados.

- Figura 78, Actuador Bombilla
- Figura 79, Actuador Puerta
- Figura 80, Actuador Alarma
- Figura 81, Sensor Sensor de Luz
- Figura 82, Sensor Humo
- Figura 83, Sensor Presencia

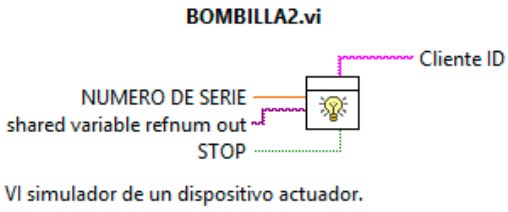


Figura 78: Actuador Bombilla

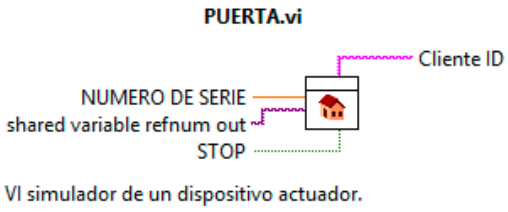


Figura 79: Actuador Puerta

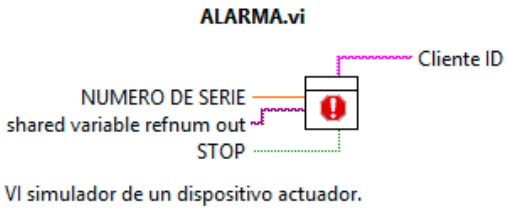


Figura 80: Actuador Alarma

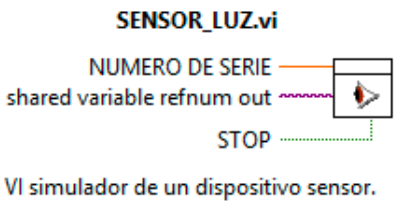


Figura 81: Sensor Sensor_Luz

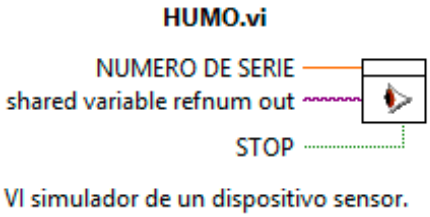


Figura 82: Sensor Humo

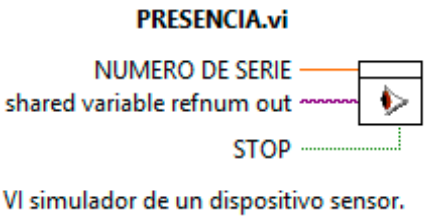


Figura 83: Sensor Presencia

6.4.2 Diseño de interfaz de usuario

Con el proyecto de interfaz de usuario, se pretende poder actuar sobre los dispositivos simulados, tanto sobre los sensores como los actuadores. Para ello, como se ha comentado anteriormente se emplean las variables globales de *Labview*.

Se crea una librería de variables globales asociada al proyecto. Todas estas variables se definen como Booleanas y tienen permisos de acceso de Lectura y Escritura. En la Figura 84 se muestra el contenido de la librería PROYECTO.

Cada variable se emplea en un proyecto diferente simulando dispositivos, excepto la variable DESCONEXIÓN que está vinculada a todos los dispositivos, para poder asi facilitar la desconexión de todos los dispositivos al finalizar la simulación.

El uso de variables globales es muy cómodo, dado que nos permite manejar indicadores y controles que se encuentran en diferentes VIs, actuando desde un solo VI.

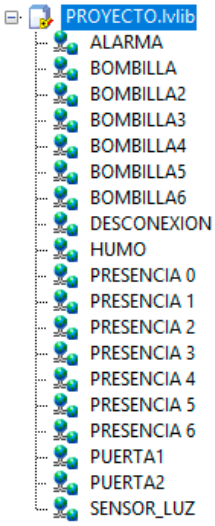


Figura 84: Librería Proyecto

Si se desea ampliar dispositivos, sólo hay que crear una variable global nueva en este proyecto, configurar en el dispositivo y añadirlo en el control de la automatización del sistema.

Ahora que se han explicado las variables globales y para que se emplean, solo hay que utilizarlas para realizar el diseño de la interfaz de usuario.

En la Figura 85, se muestra el PF del interfaz de usuario que se encuentra en el VI, Proyecto.vi. En él se distinguen tres zonas:

1. Zona de luminaria.

   En esta zona se simula la automatización de la luminaria, encendiéndose las bombillas cuando es necesario, y actuando sobre los detectores de presencia y el sensor de luz.

2. Zona de incendio.

   En esta zona se simula la automatización en caso de incendio, encendiéndose la alarma, simulando el cierre de puertas y actuando sobre el detector de humo.

3. Zona de desconexión.

   Como se ha mencionado anteriormente, este control está vinculado a todos los proyectos de simulación de dispositivos y realiza la desconexión controlada de todos ellos, sin necesidad de realizarla dispositivo por dispositivo.

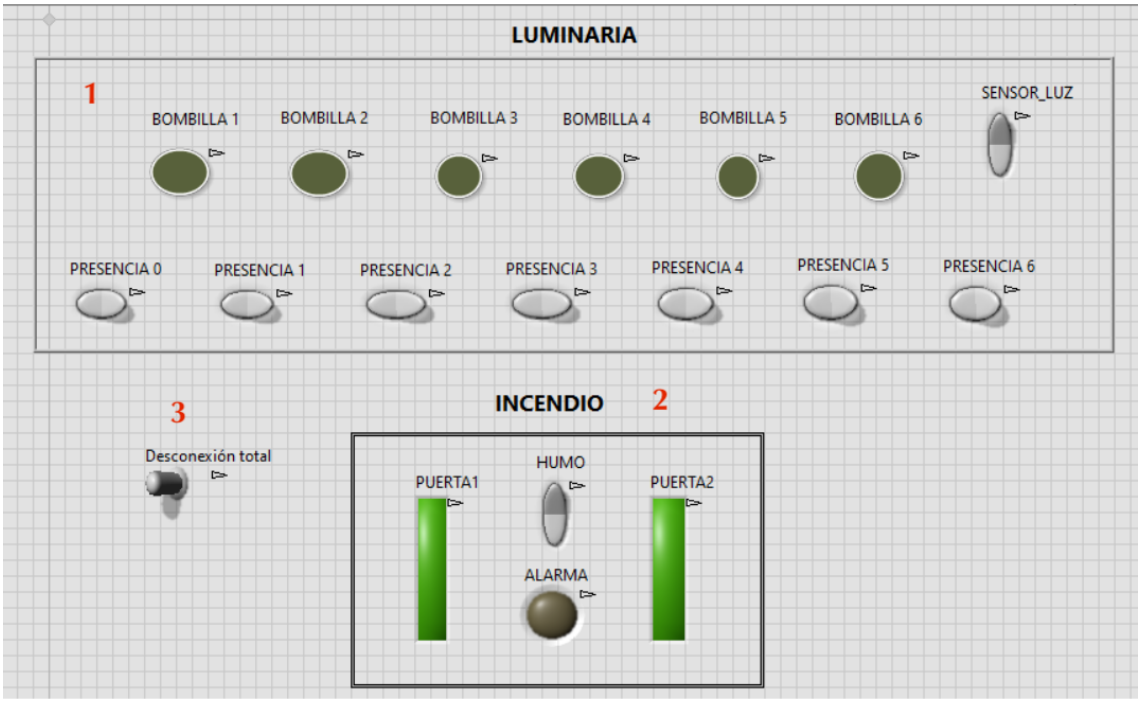


Figura 85: PF Proyecto.vi

En Figura 86 se muestra el DB de la interfaz de usuario, en el que se observa un bucle While con el control de desconexión total conectado a la finalización del bucle y los indicadores y controles asociados a las variables globales sin interconectar.

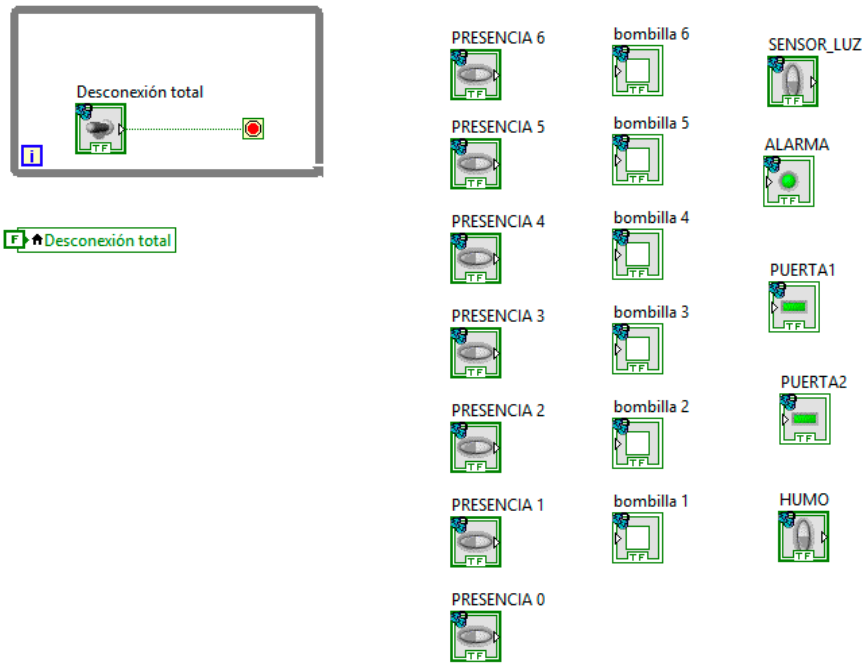


Figura 86: DB Proyecto.vi

Como conclusión de este apartado, debe quedar claro que el interfaz de usuario no tiene inteligencia, y por lo tanto no funcionaría por sí solo. Mientras que los simuladores de dispositivos sí.

El interfaz de usuario se emplea únicamente como herramienta para introducir parámetros a los simuladores de dispositivos, y reproducir que está sucediendo en ellos de manera global y en una sola pantalla.

Los VI que tienen el grueso del trabajo son los simuladores de dispositivos. Ellos se encargan de suscribirse a *Topics* y publicar mensajes en los *Topics* correspondientes. Todo ello, en función de los parámetros recibidos a través del interfaz de usuario.

Si no hubiera interfaz de usuario, la automatización seguiría siendo posible y funcionaria todo correctamente. La diferencia radicaría en que, en vez de observar un solo panel donde todos los dispositivos están agrupados, habría que comprobar el correcto funcionamiento de la aplicación de control en varios paneles aislados, uno por dispositivo.

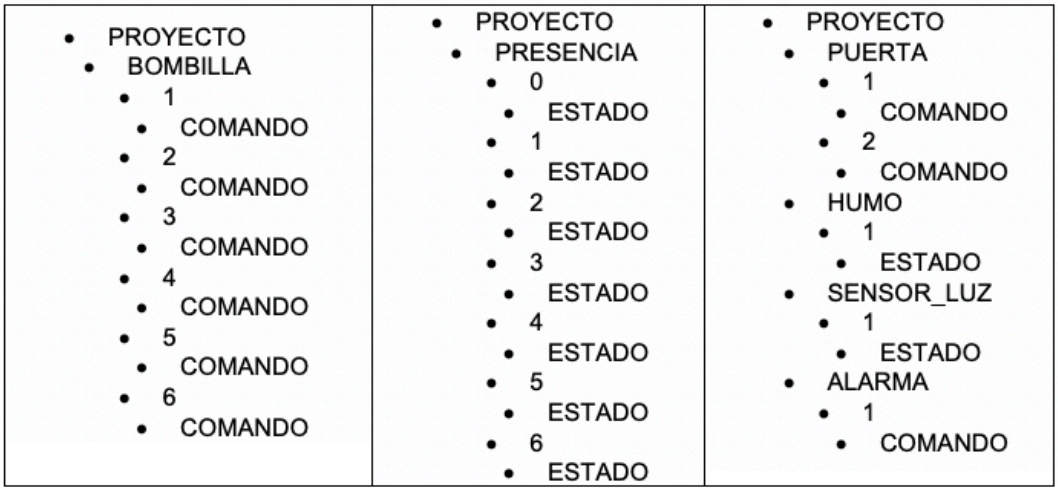
6.5 Diseño del programa de control

Antes de realizar el programa de control. Hay que analizar qué es lo que se desea automatizar y pensar cuál es la mejor manera de abordar el problema.

Analizando las especificaciones del diseño, explicadas en apartados anteriores, se observa que, en el caso de estudio, hay 9 dispositivos actuadores y 9 dispositivos sensores. Los dispositivos sensores publican en unos *Topics* y los dispositivos actuadores se suscriben a unos *Topics*.

Recordamos que los dispositivos actuadores son: Bombillas, Puertas y Alarmas; y que los dispositivos sensores son: Presencias, Humos y Sensor de Luz.

Teniendo clara la función de cada uno de los dispositivos, lo primero que se debe plantear es el árbol jerárquico del sistema. El árbol quedaría como se detalla a continuación:



6.5.1 Funcionamiento básico de conexión y lectura de mensajes publicados

El programa de control es un cliente más dentro de la comunicación, por lo tanto, debe tener las capacidades de cualquier otro cliente MQTT. Se debe poder conectar con el *Broker* e intercambiar mensajes con él.

En apartados anteriores, se ha estudiado cual es el comportamiento de los dispositivos sensores, cuyos clientes asociados envían mensajes PUBLISH, y el de los dispositivos actuadores, cuyos clientes asociados se suscriben a unos tTpics y leen los mensajes PUBLISH.

En este caso, el cliente Logica, tiene capacidades de escritura, enviando mensajes PUBLISH, y capacidades de lectura, suscribiéndose a *Topics* y leyendo los mensajes PUBLISH asociados a cada *Topic*.

En la Figura 87, se observa la parte de conexión con el *Broker* del cliente Logica.

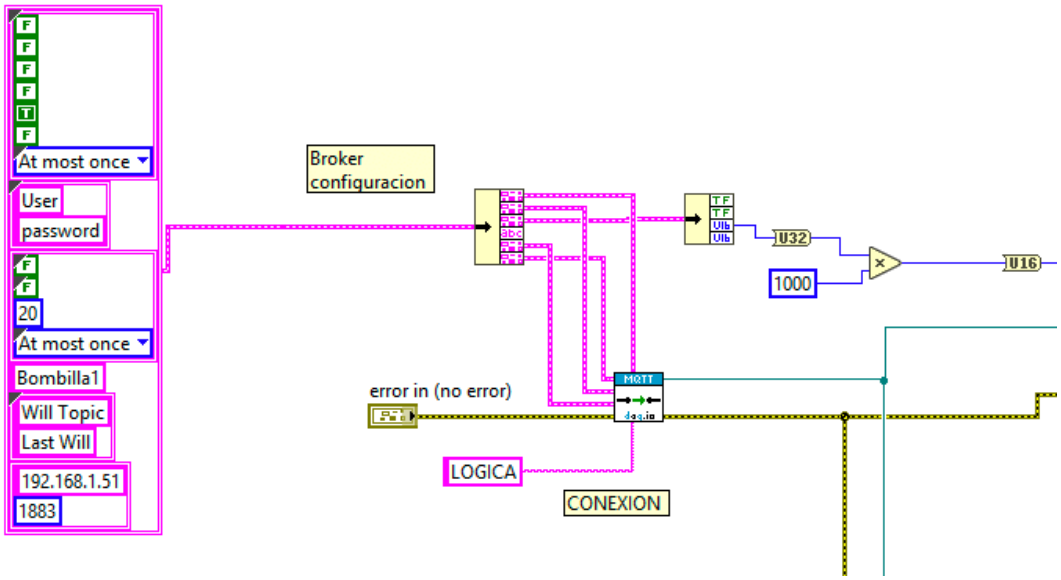


Figura 87: Conexion cliente LOGICA

En la Figura 88, se observa la parte de gestión de mensajes recibidos del VI Connect_alive2.vi. Como se ha explicado en apartados anteriores, primero se tiene que diferenciar en el tipo de mensaje que se debe gestionar. Los mensajes que este cliente debe ser capaz de gestionar son: CONNACK, PUBACK, SUBACK, PINGRESP, PUBLISH, Reserved.

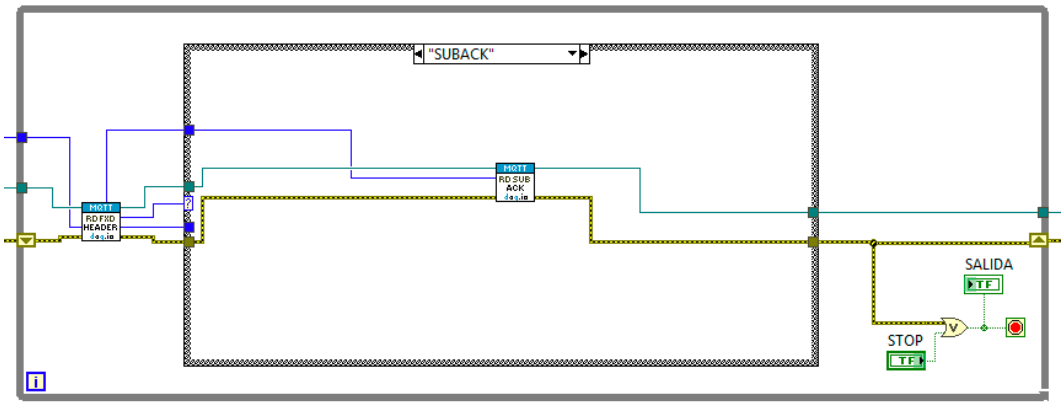


Figura 88: GESTIÓN de mensajes cliente LOGICA

- CONNACK

En la Figura 89 se observa la gestión del mensaje CONNACK. En ella se realiza la lectura del mensaje y se activa el indicador de conexión si la conexión se ha establecido con éxito. También se aprovecha y se realiza la suscripción a todos los *Topics* con un QoS 0. En la Figura 90 se muestra la lista de *Topics* suscritos.

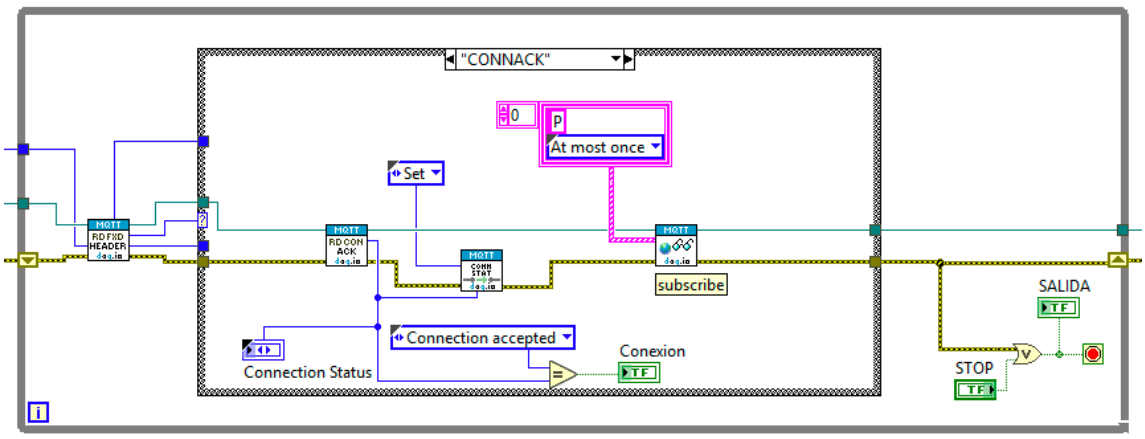


Figura 89: GESTIÓN mensaje CONNACK cliente LOGICA

TOPIC
PROYECTO/PRESENCIA/0/ESTADO
PROYECTO/PRESENCIA/1/ESTADO
PROYECTO/PRESENCIA/2/ESTADO
PROYECTO/PRESENCIA/3/ESTADO
PROYECTO/PRESENCIA/4/ESTADO
PROYECTO/PRESENCIA/5/ESTADO
PROYECTO/PRESENCIA/6/ESTADO
PROYECTO/SENSOR_LUZ/1/ESTADO
PROYECTO/HUMO/1/ESTADO

Figura 90: Lista de Topics suscritos cliente LOGICA

- PINGRESP

  Se propaga la entrada a la salida.

- SUBACK

  Lee el mensaje SUBACK. El estudio de este caso, se explicó en apartados anteriores.

- Reserved

  Este caso es para cuando pasa el tiempo establecido en el Keep Alive. El estudio de este caso, se explicó en 6.4.1.

- PUBACK

  Lee el mensaje y responde en función del QoS que tenga asociado el mensaje PUBLISH publicado. En el caso de estudio, las publicaciones se realizan con QoS 0, por lo tanto, no sería necesario gestionar el caso. Pero se decide implementar la posibilidad de tener otros QoS para posteriores usos. En la Figura 91 se muestra la gestión del mensaje.

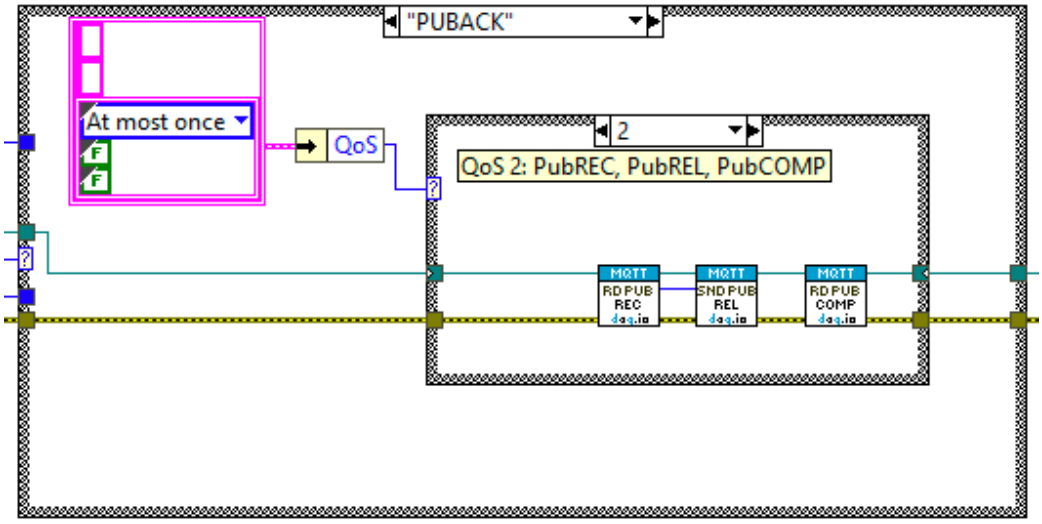


Figura 91: Gestión mensaje PUBACK cliente LOGICA

- PUBLISH

  Se encarga de la lectura de los mensajes PUBLISH recibidos. Este caso se explicará en el capítulo 6.5.2.

6.5.2 Recepción de datos de los sensores: variables intermedias

Para realizar el control del sistema de automatización, se debe poder recibir lo que publican los sensores y mandar una consigna a los actuadores. Para conseguir este propósito, se emplean unas variables intermedias que se utilizarán en la lógica de control.

Los sensores mandan un mensaje PUBLISH a su *Topic* asociado cuando se detecta un evento. En la Figura 92 se muestra la gestión de los mensajes PUBLISH.

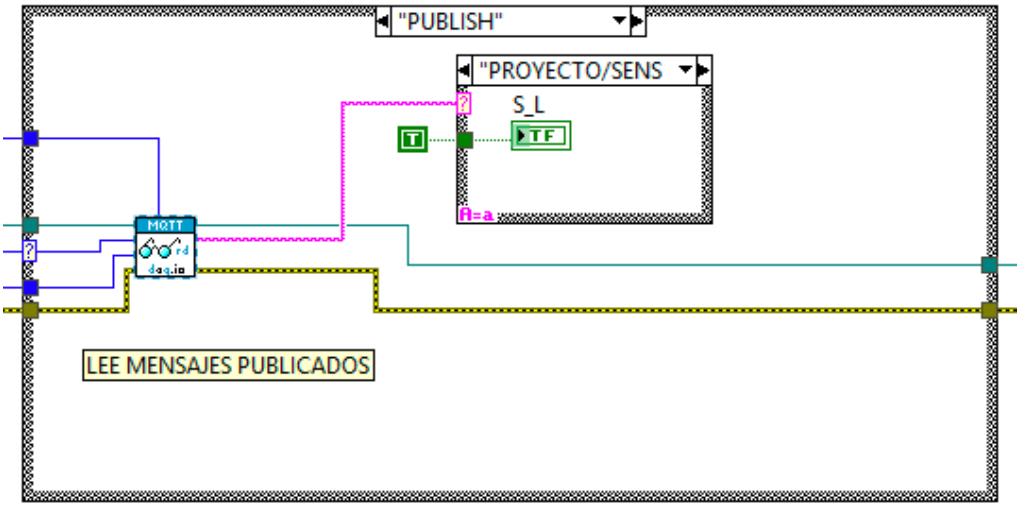


Figura 92: GESTIÓN mensaje PUBLISH cliente LOGICA

Lo realmente interesante de la lectura del mensaje es el *Topic* del mismo, dado que dependiendo del *Topic*, modificamos el valor de la variable intermedia asociada, hay una variable intermedia por cada sensor.

En la Figura 93 se muestra una tabla con los *Topics* y las variables intermedias que modifican.

TOPIC	VARIABLE
PROYECTO/PRESENCIA/0/ESTADO	P_0
PROYECTO/PRESENCIA/1/ESTADO	P_1
PROYECTO/PRESENCIA/2/ESTADO	P_2
PROYECTO/PRESENCIA/3/ESTADO	P_3
PROYECTO/PRESENCIA/4/ESTADO	P_4
PROYECTO/PRESENCIA/5/ESTADO	P_5
PROYECTO/PRESENCIA/6/ESTADO	P_6
PROYECTO/SENSOR_LUZ/1/ESTADO	S_L
PROYECTO/HUMO/1/ESTADO	H

Figura 93: Variables intermedias asociadas a sensores

También se utilizan otras variables intermedias para realizar el control de la automatización, estas variables se muestran en la Figura 94, donde se realizan las inicializaciones al entrar en el VI Connect_alive2.vi.

Al arrancar el VI por primera vez, se resetean las variables temporales poniéndolas a cero con una fecha muy anterior a la actual para evitar posibles fallos. Las variables Booleanas intermedias asociadas a cada sensor, se

inicializan a False para que no puedan guardar un dato almacenado de ejecuciones anteriores.

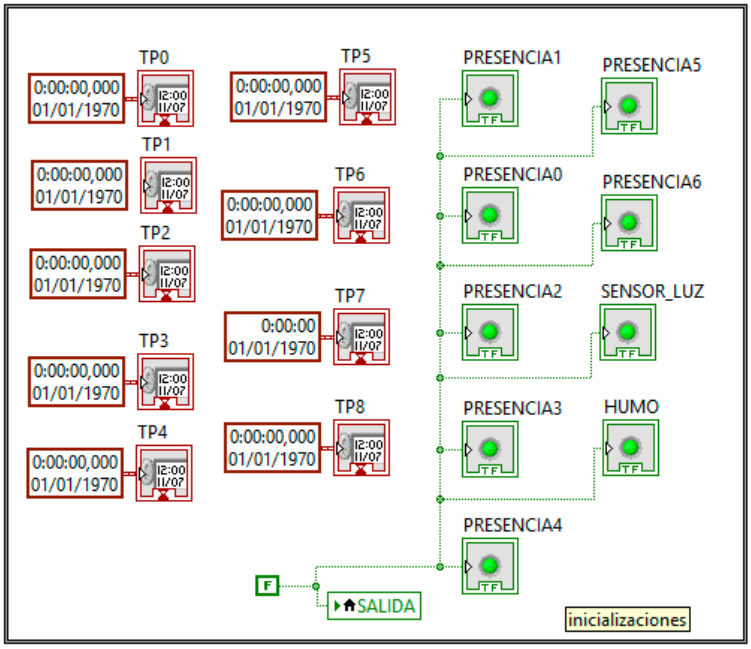


Figura 94: Inicialización variables intermedias connect_alive2.vi

6.5.3 Control de automatización

Dentro del control para la automatización del sistema, hay dos bloques o partes diferenciadas, una de ellas es la encargada de la no detección de eventos por parte de los sensores, y la otra parte es la encargada del control.

Los dispositivos sensores sólo publican cuando detectan un evento, por lo tanto, hay que saber cuándo dejan de detectar ese evento.

Para ello, se decide implementar en *Labview* una manera para comprobar si transcurrido un tiempo, sigue existiendo el evento o ha dejado de existir, de este modo, podemos realizar un control más exacto.

Para ello, se emplea el VI “Estado_Presencia” cuyas entradas y descripción se muestran en Figura 95.

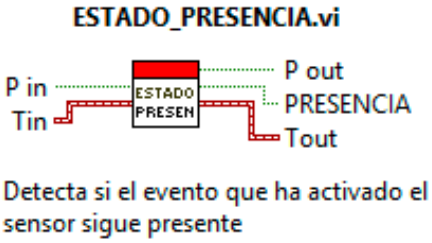


Figura 95: Icono y descripción estado_presencia.vi

La estrategia de funcionamiento que se decide implementar es la que se muestra en el diagrama de flujo que se muestra en Figura 96. Se valora si la variable de entrada es nula. Si no es así, se toma el tiempo actual para la variable de tiempo de salida y la salida Po se pone a 0. Si es así, se pasa el valor de la variable de tiempo de entrada a la variable de tiempo de salida. Después se compara el tiempo actual con el valor de la variable de tiempo de salida y si es menor o igual de dos segundos, el valor de la variable de salida es positivo. Si es mayor, el valor será negativo.

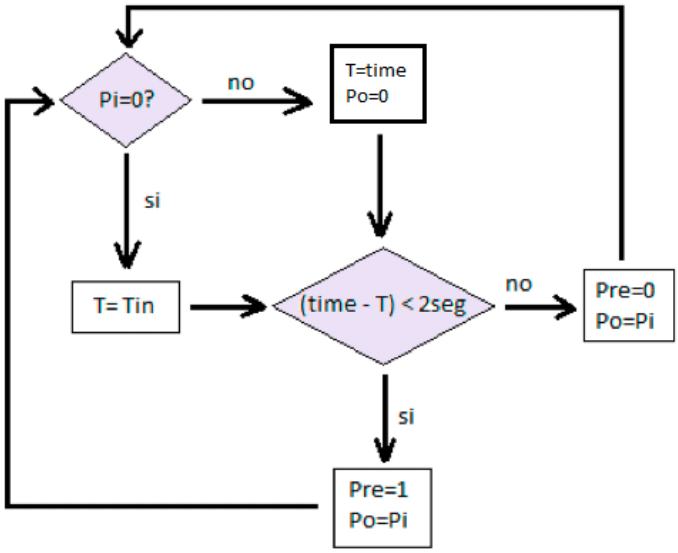


Figura 96: Diagrama flujo de Estado presencia

Como se observa en el diagrama de flujo, es necesario que la ejecución sea secuencial, por lo tanto, en *Labview* hay que implementar la programación en una estructura secuencial. El DB de Estado_Presencia.vi se muestra en Figura 97.

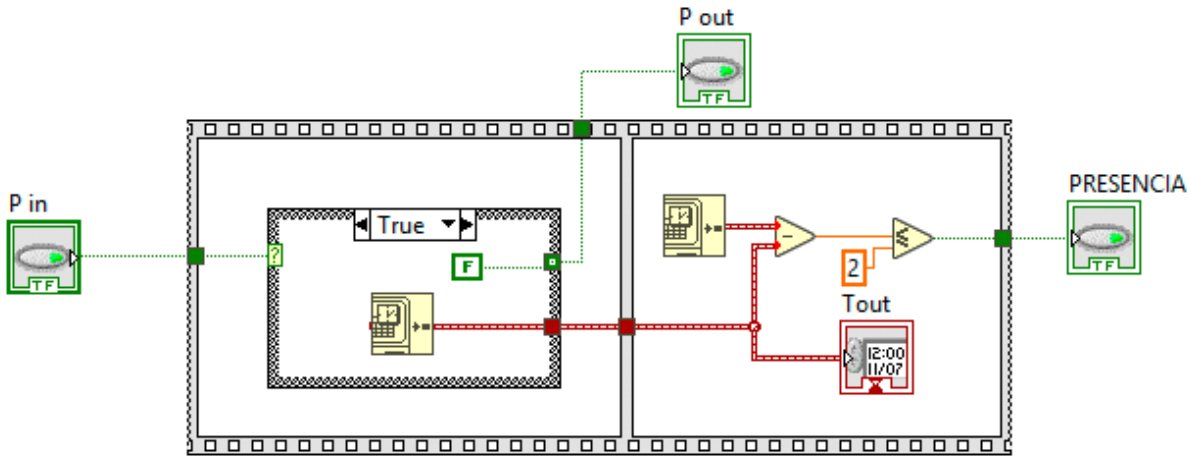


Figura 97: DB estado presencia.vi

Este proceso se debe realizar para cada uno de los dispositivos sensores, pasando como parámetros de entrada, las variables intermedias explicadas en 6.5.2. En la Figura 98 se muestra este bloque de la parte de control. Se puede observar que la variable Pin y Pout de cada dispositivo es la misma variable intermedia. Con esto se consigue que al entrar en el subVI estado_presencia, ponemos a 0 el valor de la variable que se escribe a la llegada del *Topic*, y comprobar si realmente se ha producido un evento o por el contrario ya no existe evento.

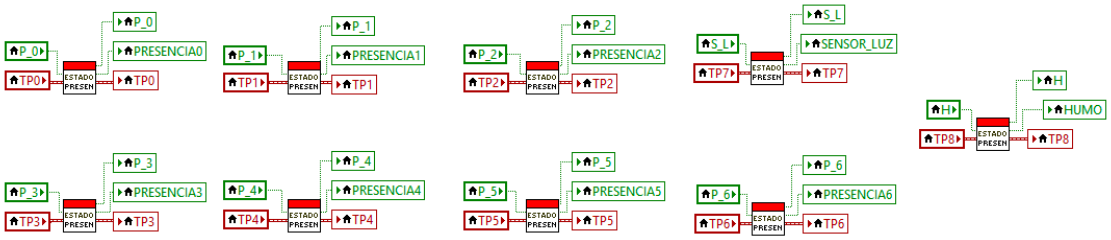


Figura 98: Parte detector de eventos connect_alive2.vi

El control de la automatización se deja a libre elección del programador mientras cumpla las especificaciones del diseño. Con respecto a la luminaria, las especificaciones del sistema descritas en 6.1, indican que, en ausencia de luz exterior, las bombillas se enciendan según presencia.

Se ha decidido que tenga el funcionamiento mostrado en la Figura 99, de modo que mientras alguno de los sensores adyacentes tenga detectado un evento, la Bombilla debe estar encendida y permanecer en ese estado hasta transcurridos dos segundos desde la finalización del evento.

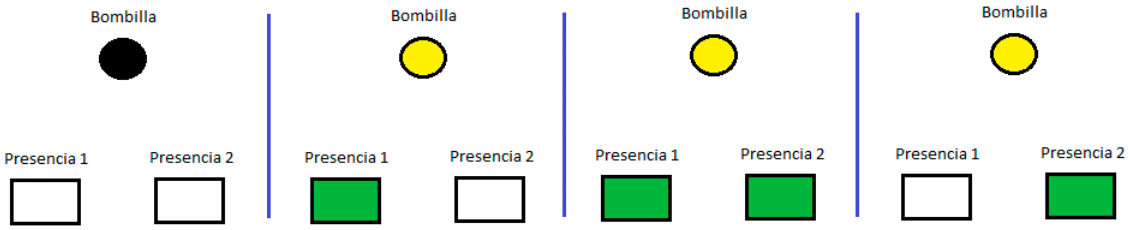


Figura 99: Casos de funcionamiento luminaria

Como se puede observar, es el funcionamiento de una puerta OR, por lo tanto, el control es muy básico. A este funcionamiento hay que añadir que solo se encenderán en ausencia de luz exterior, es decir, cuando el Sensor de luz este activo.

Para finalizar, se automatiza el control de detección de humo, cuya especificación indica que, en presencia de humo, se cierren las Puertas de emergencia, se active la Alarma y se enciendan las Bombillas, independientemente de la luz exterior.

En la Figura 100 se muestra la lógica de control y en la Figura 101 se muestra el control en su totalidad: la lógica de control y el detector de eventos de cada dispositivo.

El bucle de control se trata de un proceso no restrictivo, por lo tanto, para no poner en compromiso la ejecutabilidad del sistema, se decide implementarlo en un bucle temporal cada 20 ms, haciendo más lenta la ejecución, pero lo suficientemente reactiva para que no haya problemas.

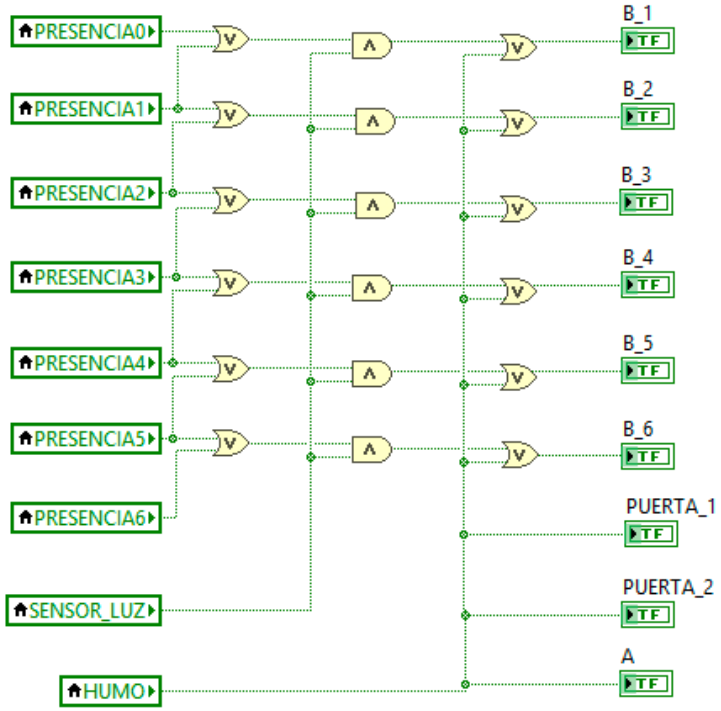


Figura 100: Control luminaria e incendio

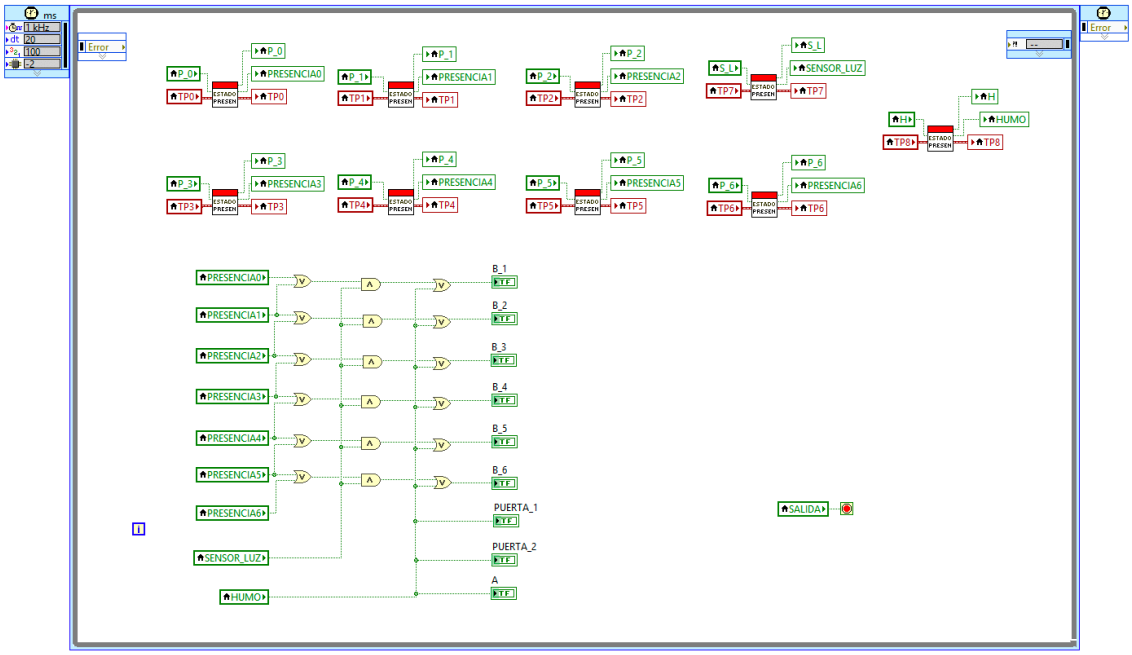


Figura 101: Bucle de control cliente LOGICA

6.5.4 Envío de datos a los actuadores

El envío a los dispositivos actuadores se realiza cuando se detecta el evento y cuando se descubre que el evento ha dejado de existir. Para ello, se realiza el envío de un mensaje PUBLISH sólo si hay un cambio en la variable de salida de la lógica de control.

Como sucedía en con el bloque de control, se decide evaluar la necesidad de envío de mensajes cada 20 ms, tiempo suficiente para que el sistema sea reactivo sin comprometer la ejecutabilidad del sistema.

Como estamos simulando 9 dispositivos, dentro del bucle temporalizado, ejecutamos un bucle FOR que contiene una ejecución por cada dispositivo simulado. Dependiendo de la ejecución del bucle en la que nos encontramos, se comprueba la necesidad de envío de cada dispositivo. Un ejemplo de esto se muestra en la Figura 102, donde se muestra el ejemplo de una de las bombillas. En este caso se trata del cliente BOMBILLA2.

En esta parte del bloque, se decide a qué tipo de cliente va dirigido el mensaje, y se propaga el número de ejecución del bucle para posteriormente especificar cuál de ellos es. También se propaga el valor de salida de la lógica de control asociada al cliente y el resultado de la comprobación de cambio de estado.

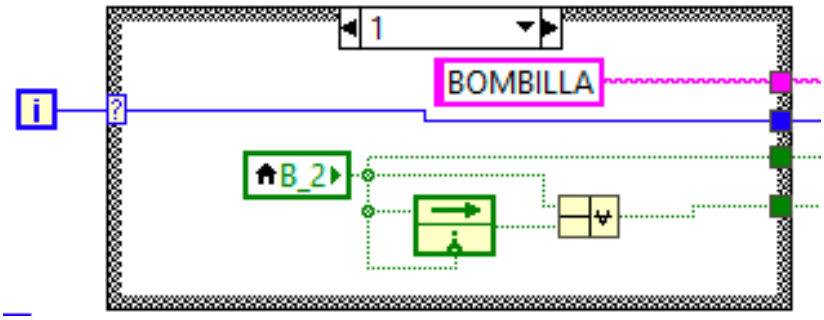


Figura 102: Bucle envío: "Deteccion de cliente" en cliente LOGICA

En la Figura 103 se observa la segunda parte del bucle, si se ha producido un cambio de estado en la variable de salida de la lógica de control, se envía el mensaje PUBLISH habiéndose seleccionado el cliente dentro del tipo de clientes, e introducido el *Topic* y el contenido del mensaje.

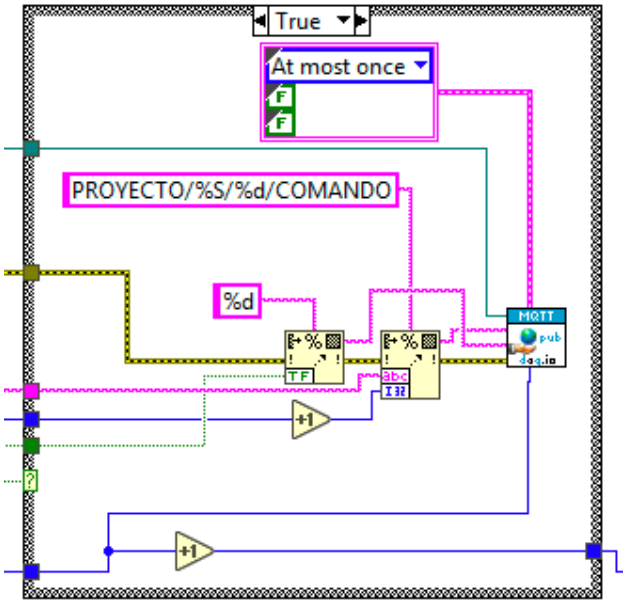


Figura 103: Bucle envío: "envío de mensaje" en cliente LOGICA

Por último, en la Figura 104 se muestra el bucle de envío completo.

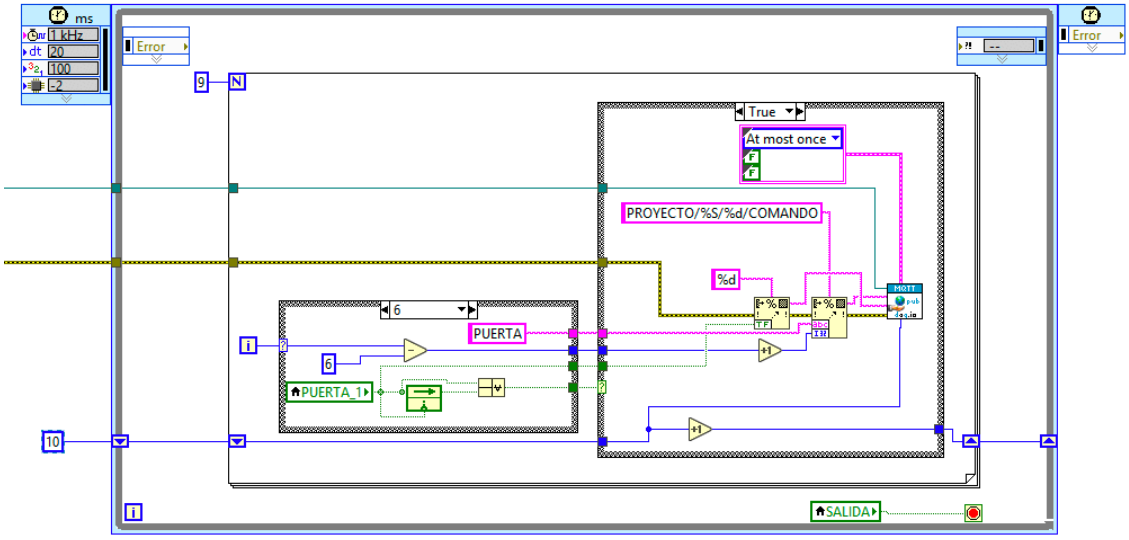


Figura 104: Bucle Envío completo en cliente LOGICA

6.6 Resultados y conclusiones

Para finalizar el caso práctico, se realiza la simulación de todo el sistema completo: *Broker* y clientes.

Lo primero que se debe realizar es desplegar el *Broker* MQTT para que los dispositivos y la lógica de control puedan intercambiar los mensajes a través de él. Una captura de pantalla del *Broker* desplegado es la mostrada en la Figura 105.

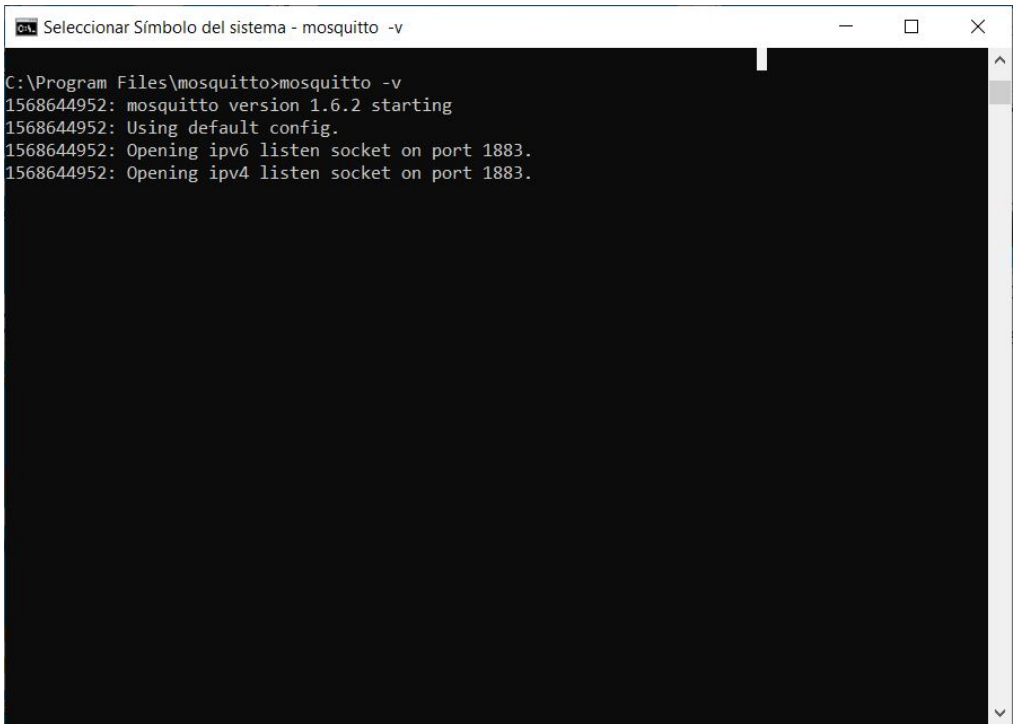


Figura 105: Broker desplegado

Antes de desplegar los clientes MQTT, hay que tener en cuenta algunos factores como:

1) Inicialización de variables globales:

Hay que asegurarse que las variables globales están inicializadas con su valor correspondiente, si la variable DESCONEXIÓN está inicializada con un True, tendremos errores de conexión con el *Broker*, tanto en el interfaz de usuario como en los dispositivos. Esta comprobación se puede realizar en la herramienta "NI Distributed System Manager".

En Figura 106 se muestra cómo deben estar inicializadas las variables. De este modo, todos los sensores se inicializan sin eventos y los actuadores se inicializan desconectados.

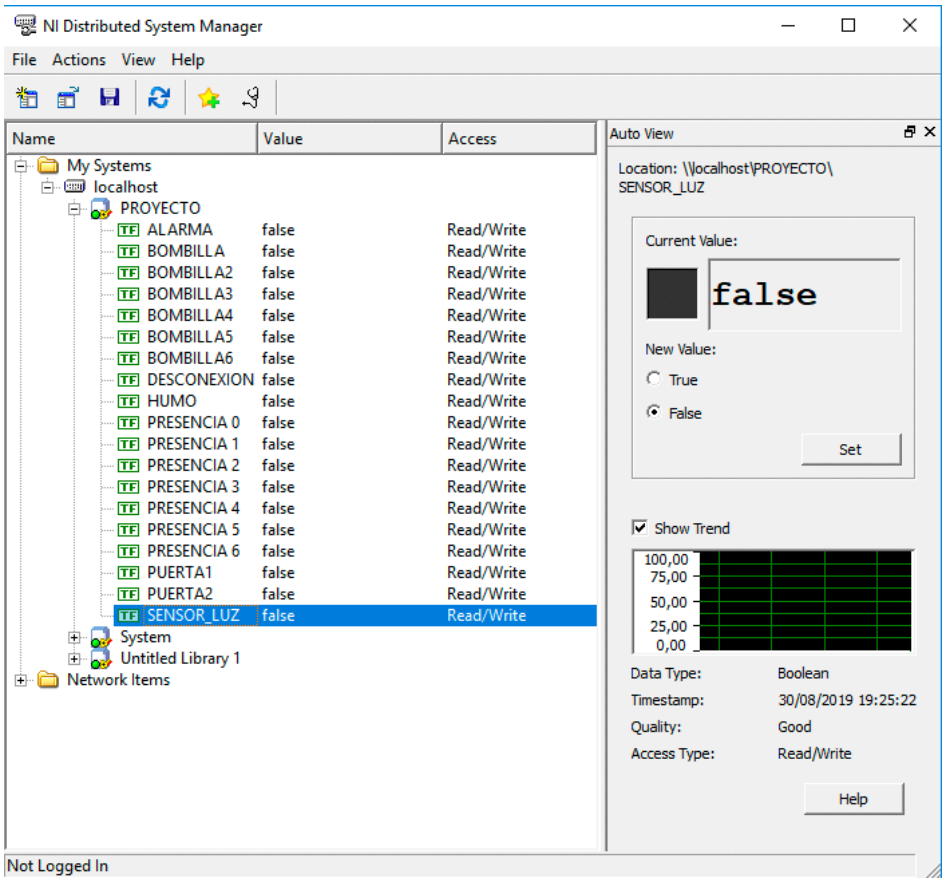


Figura 106: NI Distributed System Manager

2) Desconexión anterior:

Hay que tener en cuenta que si los clientes no se han desconectado del *Broker* anteriormente de una manera correcta y la desconexión se ha producido por falta de comunicación, también habrá errores en la conexión y habrá que lanzar la ejecución del cliente dos veces para que se realice correctamente la conexión con el *Broker*.

Una vez que se han tenido en cuenta las observaciones anteriores, se despliegan los clientes MQTT en el siguiente orden:

1. Herramienta MQTTBox
2. Cliente Logica,
3. Interfaz de usuario
4. Resto de clientes

Como se ha explicado en el capítulo 6.4.2, el interfaz de usuario no es un cliente propiamente dicho, pero es el proyecto que nos permite interactuar con todos los clientes desde una misma ubicación sin tener que cambiar de ventana.

Una vez desplegado el sistema completo, se realizan diferentes pruebas de funcionamiento.

Entre esas pruebas se encuentran:

1) Comprobar el correcto funcionamiento de las variables globales, observando si los interruptores de presencia y los indicadores del interfaz de usuario se comunican con los dispositivos.

   En las siguientes Figuras se muestra el resultado de la prueba descrita anteriormente tomando como ejemplo la variable global asociada al sensor PRESENCIA_0. Se aprecia el dispositivo a la izquierda de la Figura y el interruptor PRESENCIA_0 correspondiente al interfaz de usuario.

   En la Figura 107 se observa que el interruptor “Publica mensaje” está desactivado en el dispositivo y el interruptor asociado en el interfaz de usuario también se encuentra desactivado.

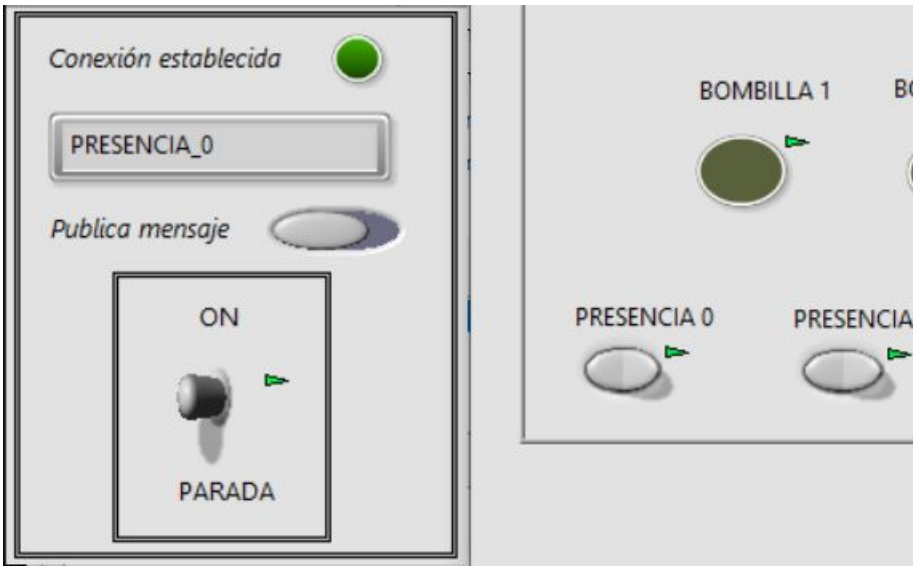


Figura 107: Sensor PRESENCIA 0 desactivado

En la Figura 108 se observa que el interruptor “Publica mensaje” está activado en el dispositivo y el interruptor asociado en el interfaz de usuario tambien se encuentra activado.

Cuando se realiza un cambio de estado en el interfaz de usuario, también se realiza un cambio de estado en publica mensaje del dispositivo. Esto es extensible a todos los dispositivos simulados en este caso de uso, tanto los dispositivos Actuadores, como los dispositivos Sensores.

A su vez, para facilitar la correcta desconexión de todos los dispositivos, en la aplicación de usuario, hay una variable global unida a todos los dispositivos que hace que se solicite la desconexíon de todos a la vez, evitando la tarea de ir dispositivo, por dispositivo.

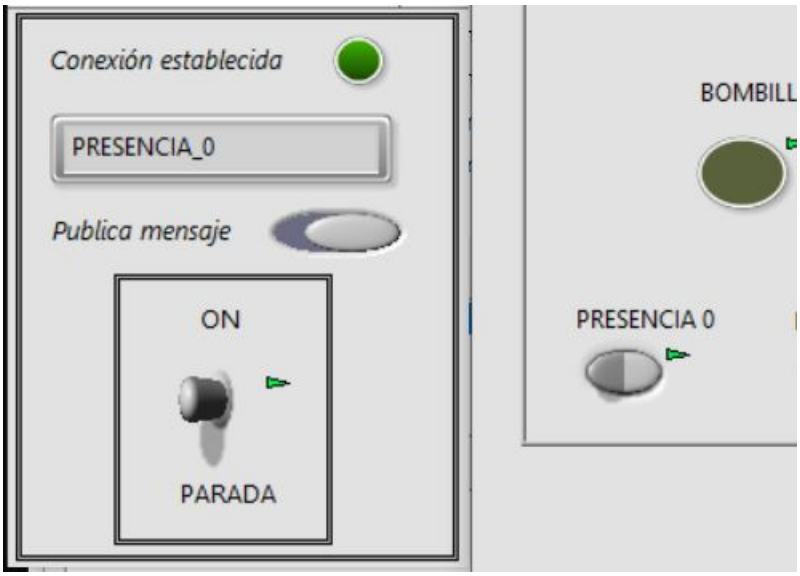


Figura 108: Sensor PRESENCIA 0 activado

2) Comunicación de los clientes con el *Broker* MQTT, tanto la suscripción como la publicación de *Topics.* Para ello, se conectan dos dispositivos al *Broker*, de los cuales: uno realiza una publicaion en un *Topic* y otro se suscribe al mismo *Topic*.

   En las siguientes Figuras se muestra el resultado de está prueba, donde se puede apreciar la existencia de comunicación entre *Broker* y dispositivos, asi como el intercambio de mensajes.

   En la Figura 109, se observa:

   (a) Como el dispositivo PRESENCIA_0 está conectado y mandando peticiones PINREQ

   (b) Otro dispositivo solicita conectarse con el *Broker*. Este nuevo dispositivo es la herrramienta MQTTBox.

(c) MQTTBox, solicita suscribirse al *Topic* “PROYECTO/PRESENCIA/0/ESTADO”. Ese *Topic*, es en el que publica el dispositvo PRESENCIA_0.

(d) PRESENCIA_0 realiza cuatro publicaciones hacia el *Broker*, y este realiza cuatro publicaciones hacia MQTTBox.

En la Figura 110, se muestran tres de los cuatro mensajes enviados por el dispositivo PRESENCIA_0 y recibidos en MQTTBox a través del *Broker* MQTT.

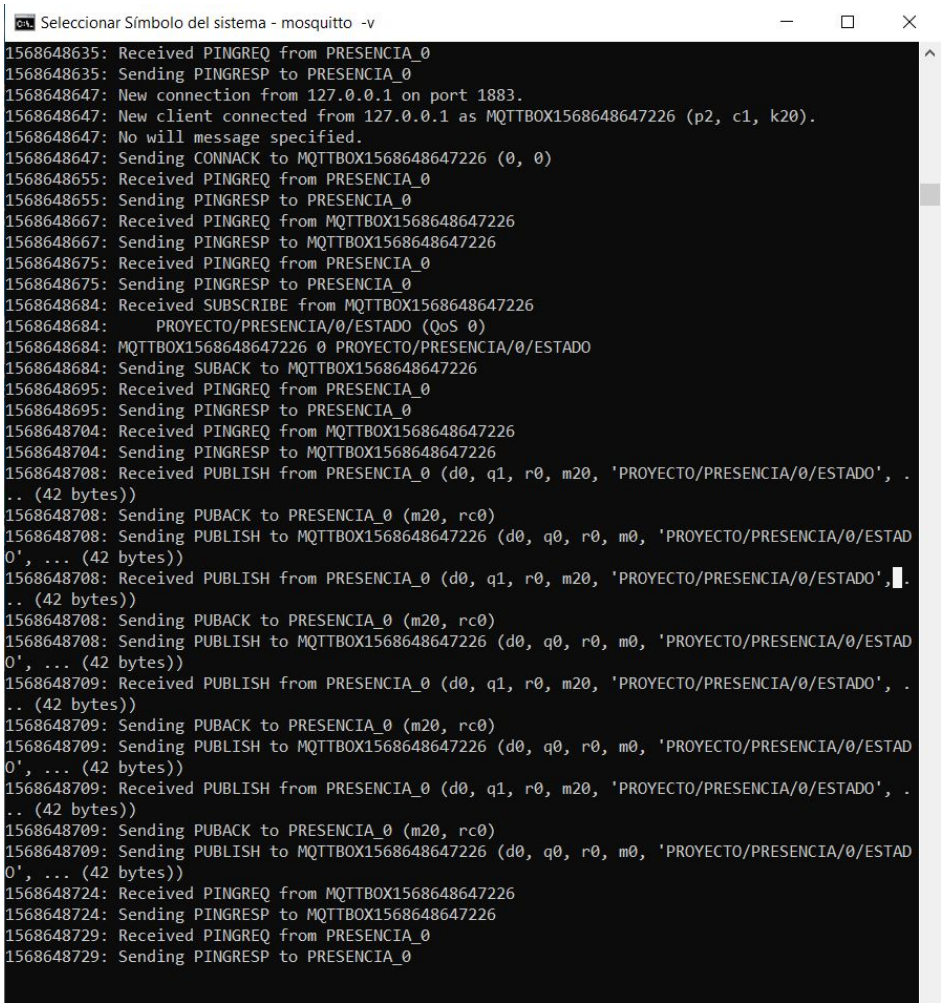


Figura 109: Intercambio de mensajes con el Broker

Figura 110: MQTTBox suscrito a PROYECTO/PRESENCIA/0/ESTADO

3) Comprobación del sistema completo, para ellos se simula:
   a) el paso de personas a lo largo de un pasillo, tanto con luz exterior como sin ella.
   b) La activación del sensor de humo.

   Para mostrar los resultados de está prueba seria necesario mostrar el funcionamiento completo en un video o una simulación en tiempo real. La gran cantidad de información que se recopila en está prueba hace que sea muy complicado poder exponerla en texto en este libro.

7 Conclusiones y trabajo futuro

Las modificaciones sobre la librería LVMQTT para *LabView* para conectar un dispositivo con un *Broker* MQTT han posibilitado la interacción de distintas suscripciones y publicaciones en paralelo.

Esto hace que la librería sea mucho más abierta a su uso con varios dispositivos de forma simultánea, dado que cada dispositivo realiza una gestión de mensajes más dinámica, sin esperar la respuesta del *Broker* a cada uno de sus mensajes.

Esta librería permite integrar elementos de instrumentación que se emplean tradicionalmente en NI con elementos del entorno IoT e IIoT. Facilitando el desarrollo y las pruebas de este tipo de elementos, pudiendose beneficiar de los bancos de pruebas, controles y funciones en general que ofrece LabView.

El caso de uso incluye y mezcla métodos de comunicación empleados tradicionalmente en NI, como son las variables globales, para interactuar entre diferentes instancias software, con comunicaciones MQTT más propias de elementos de baja carga computacional. De esta manera, se puede sustituir en cualquier momento cualquiera de los dispositivos simulados por elementos físicos reales.

El caso de uso también proporciona una visión de las posibilidades que ofrece este entorno conjunto, pudiendo monitorizar y controlar elementos de la índole de: estaciones de carga de vehículo eléctrico, sistemas de monitorización de contadores eléctricos, campos de producción de energía, automatización en la industria en general, etc.

Como trabajo futuro a este TFG, se propone:

1) Implementar una maqueta en la cual, se intercambien los dispositivos simulados en *Labview* por unos dispositivos físicos IIoT, verificando de este modo que los resultados obtenidos en este TFG serían analogos a los resultados que se obtendrian con dispositivos reales.
2) Aplicar seguridad a las librerias MQTT creadas en Labview para implementarlas en sistemas que lo requieran, dado que en el caso de uso descrito no era necesario aplicarla.
3) Realizar un estudio exhaustivo sobre los tiempos de respuesta del protocolo, pudiendo encontrar los limites de dichos tiempos para sistemas más rápidos. En este TFG no se ha realizado tal estudio dado que los tiempos de respuesta son mucho menores a los retardos intrinsecos al control.

Presupuesto de ejecución

1 Coste de material informático

Material informático (Software)				
Concepto	Precio (€)	Duración (años)	Uso (meses)	Total (€)
Microsoft Office 365	99	1	2	16,5
Labview	3451	1	3	862,75
Total, en material informático				**879,25**

2 Coste de equipos

Equipos				
Concepto	Precio (€)	Duración (años)	Uso (meses)	Total (€)
Ordenador Intel® Core™ i7-4800MQ CPU @2.7GHz 16GB	2800	4	4	233,35
Total, equipos				**233,35**

3 Coste de personal

Personal			
Trabajador	Meses	€/hora	Total (€)
Ingeniero	480	35	16.800
Total, Personal			**16.800**

4 Coste de ejecución material

Coste material informático (Software)	879,25 €
Coste de equipos	233,35 €
Coste de personal	16.800 €
Total	**17.912,6 €**

5 Gastos generales y beneficio Industrial

Se considera un 25% del coste de ejecución material el gasto producido por la utilización de las instalaciones y el beneficio industrial.

Gastos generales y beneficio industrial	**4.478,15 €**

6 Presupuesto de ejecución por contrata

Coste de ejecución material	17.912,6 €
Gastos generales y beneficio industrial	4.478,15 €
Total, de ejecución por contrata	22.390,75 €
IVA 21%	4.702.06 €
TOTAL	**27.092,80 €**

El importe total asciende a la cantidad de:

VEINTISIETE MIL NOVENTA Y DOS EUROS CON OCHENTA CÉNTIMOS.

Alcalá de Henares a 12 de Septiembre de 2019

Proyectos y programas

Adjunto a esta memoria, se encuentra la librería de VIs empleada en este TFG.

La estructura de los mismos se detalla a continuación:

1) Carpeta PROYECTO

   Contiene los VI principales para simular cada dispositivo, el proyecto de la lógica de control y el proyecto del interfaz de usuario.

   También contiene las carpetas:

   a) Actuadores

      Contiene los proyectos de los dispositivos actuadores simulados

   b) Run

      Contiene los VIs RUN y RUNP necesarios para utilizar el interfaz de usuario gracias a las variables globales

   c) Sensores

      Contiene los proyectos de los dispositivos sensores simulados

   d) LVMQTT-master

      Contiene la librería MQTT base para *Labview*, modificada para poder emplearla en el caso descrito en este TFG.

2) Carpeta LVMQTT-master (original)

   Esta carpeta contiene la librería original MQTT para *Labview* que está disponible en GitHub.

Anexo

En este Anexo se pretende profundizar en Labview, explicando tanto sus partes, como las herramientas que se pueden emplear para ayudar a la progamación y la depuracion del código, y algunos de los VIs Express que se emplean para agilizar la programación.

1 Qué es un VI, sus partes

Los programas desarrollados con *Labview* se denominan VIs, dado que originariamente se creó para controlar instrumentos. Y se pueden subdividir en tantos subVIs como sean necesarios, dependiendo de la complejidad del programa.

La idea de *Labview* es poder hacer VIs modulares que se puedan implementar en otros VIs, para no tener que realizar la programación desde cero y así agilizar el proceso de desarrollo. Para ello se crean los VIs Express, que son procesos configurables a través de un panel de propiedades permitiendo personalizar la funcionalidad del mismo.

Cuando se crea un proyecto, este puede contener tantos VIs como sean necesarios.

Cada VI consta de dos partes:

- Panel Frontal, PF: Es el interfaz del usuario, donde se puede:
  - Configurar valores de parámetros,
  - Ingresar datos solicitados en algún momento,
  - Observar valores y graficas resultantes del programa en tiempo real,
  - etc.
- Diagrama de Bloques, DB: Es el programa propiamente dicho, donde realizamos las acciones necesarias para el correcto funcionamiento del mismo.

En la Figura 111 se muestra un VI en blanco: en la parte de la izquierda observamos el PF y en la parte de la derecha el DB. Se pueden observar los siguientes botones:

1. Botones de simulación.
2. Botones de reordenación de objetos.
3. Icono
4. Panel de conectores.
5. Botones de depuración

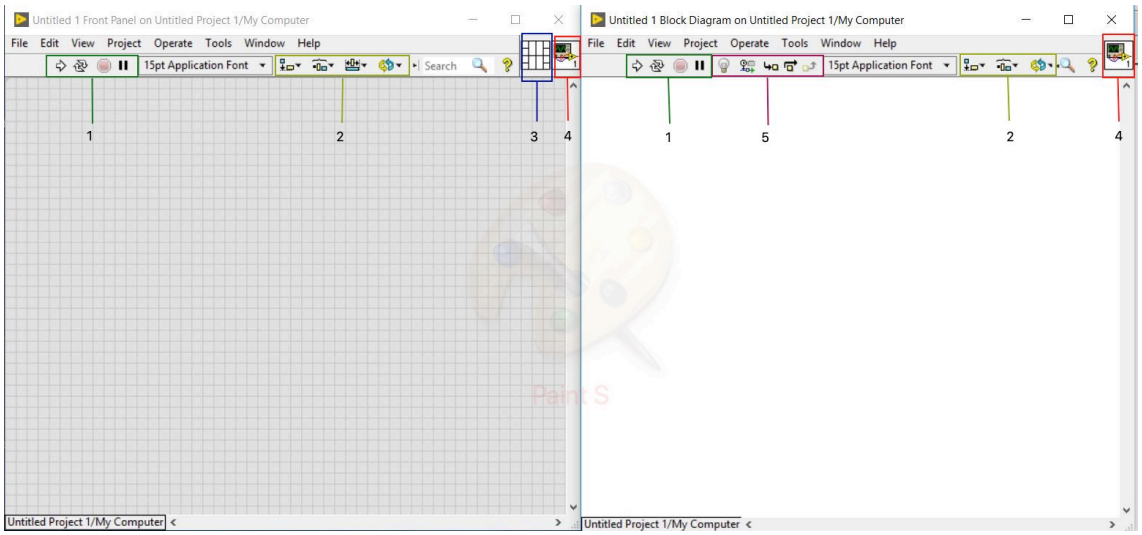


Figura 111: Partes de un VI

En el PF, se colocan los controles y los indicadores. Los controles son los elementos que se utilizan como entradas y los indicadores son los que nos muestran los datos de salida.

Tanto los controles como los indicadores pueden ser de diferentes tipos, según los datos que manejan: *booleanos, numéricos, cadenas de caracteres*.

Para más información sobre el tipo de datos empleados en *Labview*, se puede consultar [51].

Cuando se realizan operaciones matemáticas, hay que tener en cuenta que tanto controladores como indicadores, deben trabajar con el mismo tipo numérico: *float, fixed, integrer o complex*.

Si no son del mismo tipo, pueden aparecer puntos de coerción [52] y, por lo tanto, pueden existir errores de precisión en el resultado. Para solucionar este inconveniente, se pueden usar funciones de conversión en uno de los tipos para que ambos elementos sean del mismo tipo.

En la Figura 112 se muestran algunos ejemplos de librerías de controles e indicadores. Para obtener la Paleta de Control, la podemos obtener desde el menú View, tanto en el DB como en el PF.

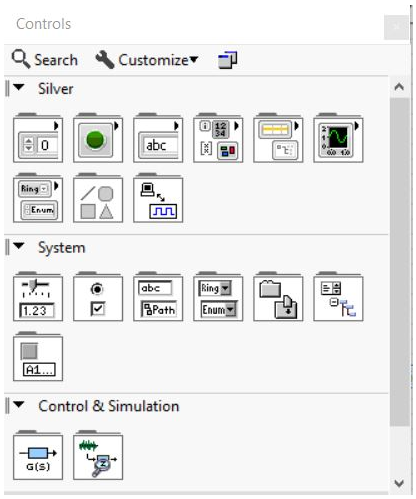


Figura 112: Paleta de control

En el DB se realiza la programación. En él aparecen:

- *los terminales*, que son los elementos asociados a los controles e indicadores que se encuentran en el PF;
- *las funciones*, que son necesarias para realizar un programa; *las notas* explicativas de cada elemento y *los cables* de interconexión de elementos.

En la Figura 113 se muestra el PF y el DB de un programa sencillo. En el PF se aprecian tres ejemplos de controles y tres indicadores asociados a esos mismos controles. En el DB se observan los elementos que componen el mismo. Remarcados en azul (1) se encuentran las funciones; en rojo (2) los terminales control; en verde (3) los terminales indicadores y los cables que hacen posible la programación.

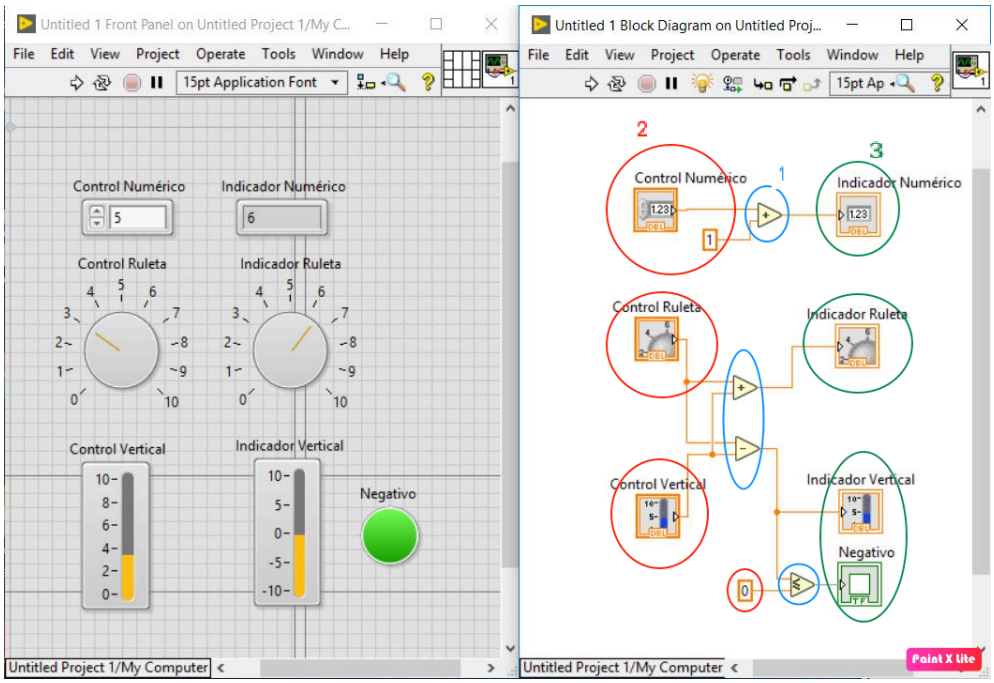


Figura 113: Partes de un programa

En la Figura 114 se muestra la Paleta de Funciones, a la cual se accede desde el menú *View* del DB. En la paleta se puede encontrar un gran número de funciones, desde funciones matemáticas simples a funciones más complejas de control o simulación de sistemas.

Figura 114: Paleta de Funciones

Para ayudar al usuario a realizar la programación existen herramientas que hacen más eficiente el programa. Estas herramientas se encuentran en la Paleta *Tools* que se encuentra en el menú *View*. En la Figura 115 se puede ver esta paleta. Entre las herramientas que se encuentran en esta paleta aparecen: *Positioning, Labering, Writing, Breakpoint, Probe*, etc.

Figura 115: Paleta Tools

También se cuenta con otras herramientas que aparecen tanto en el PF como en el DB, que ayudan con el cableado. Estas herramientas se muestran la Figura 111, y son los botones de colocación de objetos. Estos botones hacen que se ordene el programa para una mayor comprensión.

2 Metodología de programación en *Labview*

Hay que tener en cuenta que la programación que se realiza en *Labview* es secuencial y por lo tanto es importante poner a la izquierda las entradas y las salidas a la derecha, para conseguir una correcta lectura y depuración posterior.

Un ejemplo de esto se aprecia en la Figura 113, donde en la parte del DB, se ven las entradas en la parte de la izquierda, las funciones que realiza el grueso del programa en la parte central y las salidas en la parte derecha.

Como se ha visto en 3.3, la metodología de programación de los VIs es la siguiente:

- Adquisición
- Análisis
- Presentación.

A lo largo de este apartado se mencionan algunos VI Express y se explican de manera genérica los usos de cada uno de ellos, para más información, se recomienda leer la ayuda que proporciona *Labview* sobre cada uno de ellos.

2.1 Adquisición

Los VI Express más comunes son:

- *DAQ Assistant:* Adquiere datos de un dispositivo de adquisición de datos de NI, pero es necesario tener licencia específica para poder utilizarlo. Con este VI Express se pueden adquirir datos de una fuente externa realizando una sencilla configuración.
- *Simulate Signal:* Simula la entrada de datos, generando los mismos. Simula todo tipo de señales configurando la amplitud y la frecuencia.
- *Simulate Arbitrary Signal:* Simula la entrada de datos, generando señales aleatorias.
- *Read from Measurement file:* Lee los datos de medición que se encuentran en un archivo.

En la Figura 116 se muestran los VIs Express más comunes en adquisición.

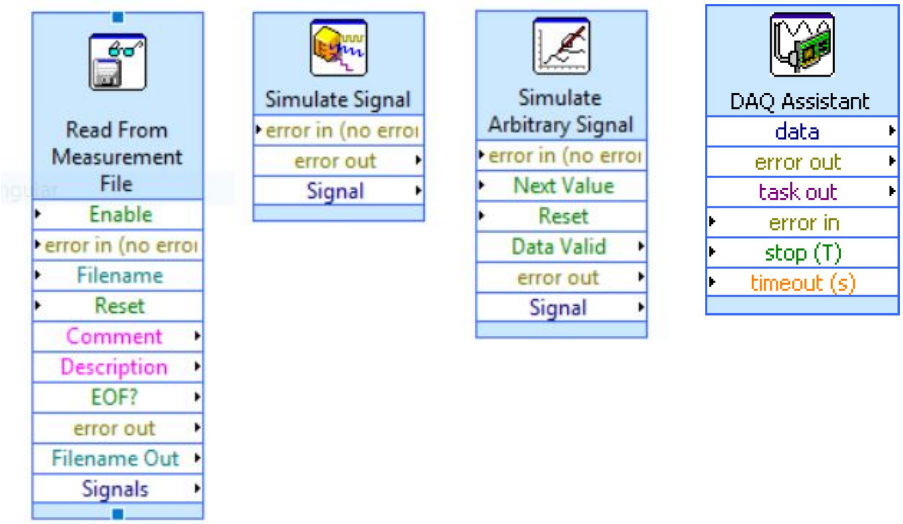


Figura 116: VI Express comunes en Adquisición de Datos

2.2 Análisis

Algunos de los VIs Express empleados en esta parte de análisis son:

- *Amplitude and level measurement:* Realiza mediciones de voltaje de una señal.
- *Statistics:* Calcula los datos estadísticos de una forma de onda.
- *Spectral measurement:* Realiza la FFT de una forma de onda.
- *Tone Measurement:* Busca la frecuencia y la amplitud de un tono.
- *Filter:* Procesa una señal a través de filtros.

En la Figura 117 se muestran los VIs Express más comunes en el análisis.

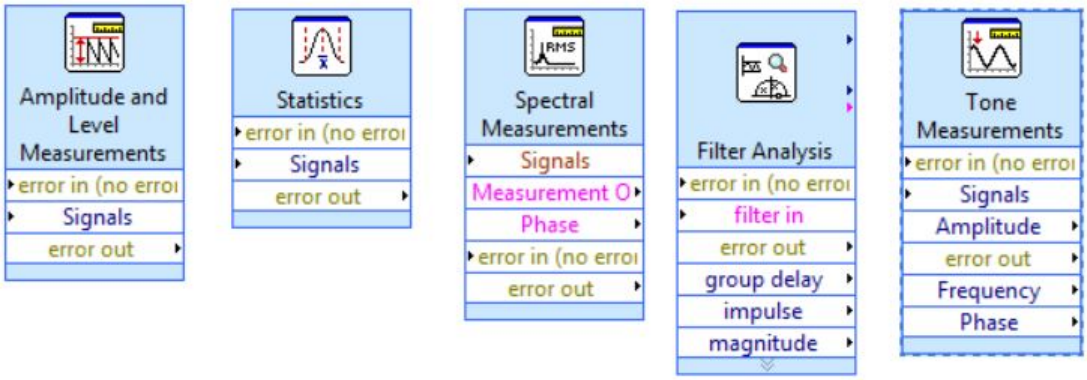


Figura 117: VI Express comunes en análisis

2.3 Presentación

Los VIs Express más comunes son:

- *Write to measurement file:* registra datos en archivos,
- *Build text:* crea un texto para mostrar en el panel de control,
- *DAQ Assistant:* genera una señal en un DAQ de NI.

En la Figura 118 se muestran los VIs Express y en la Figura 119 los indicadores más comunes en la presentación de señales.

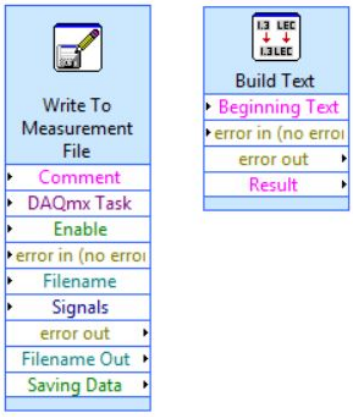


Figura 118: VI Express comunes en presentación de datos

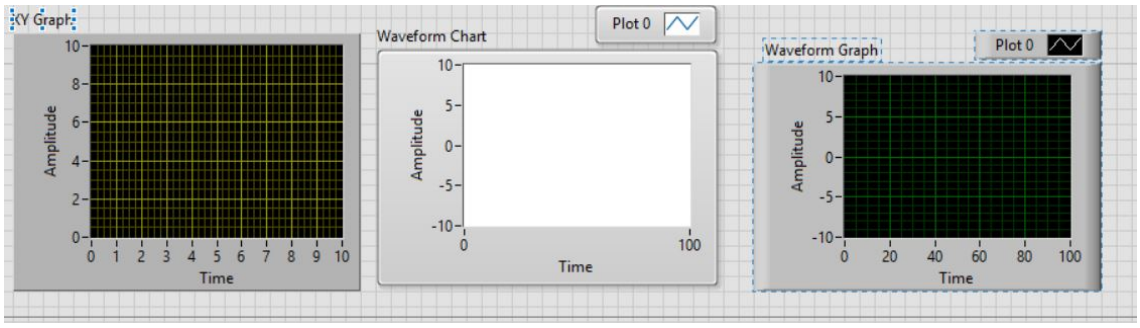


Figura 119: Indicadores más comunes para visualizar señales

3 Resolución de problemas

Tanto desde el PF como desde el DB podemos lanzar, pausar y parar la simulación. En la Figura 111 se ven los botones utilizados para la simulación y la depuración de los programas.

Lo más importante para enfrentarse a la resolución de problemas utilizando la herramienta de programación *Labview* es saber identificar los problemas más comunes, aprender a utilizarlas herramientas de depuración y realizar una correcta gestión automática o manual de los errores.

3.1 Identificación de problemas comunes

Cuando el botón de ejecución aparece roto, puede ser porque hay errores que impiden la ejecución del mismo. Lo más común puede ser, que exista un cable con extremos sin conectar, haya incompatibilidad entre tipos de datos o falta la conexión de un terminal de entrada.

Aparece una lista de errores al clicar sobre el botón roto de ejecución, cada elemento de la lista, nos lleva directamente sobre la ubicación del error.

También pueden existir errores debidos a divisiones por cero o a raíces cuadradas negativas.

3.2 Herramientas de depuración

Se deben emplear las herramientas de depuración, cuando al ejecutar el programa, no da el resultado esperado. En la Figura 120 se muestran las herramientas de depuración.

Figura 120: Herramientas de depuración

En [53], detallan el funcionamiento de estos botones; en este trabajo sólo vamos a realizar una mención a las tareas de cada uno de ellos sin entrar en detalle.

[1] *Highlight Execution:* muestra la ejecución del programa en animación para ver por dónde está el flujo de datos.

[2] *Retain Wire Values:* guarda los valores del cable en cada punto en la ejecución, para poder leerlo cuando se acerca la punta de pruebas.

[3] *Step Into:* abre un nodo y hace una pausa.

[4] *Step Over:* abre un nodo y hace pausa al siguiente nodo.

[5] *Step Out:* termina la ejecución del nodo y hace pausa.

3.3 Gestión de errores

Si está activa la gestión automática de errores, cuando sucede un error en la ejecución del VI, se resalta el lugar donde se produce el error y se abre un cuadro de diálogo, donde se muestra el error y el código del mismo para poder identificarlo.

Para desactivarla se debe conectar el terminal de salida del error al terminal de entrada de la siguiente función o subVI.

Con la gestión manual de errores, se puede controlar cuando se reporta el error sin necesidad de parar la ejecución del VI.

Para emplear la gestión manual de errores, *Labview* pone a disposición del usuario los Cluster de Error, para poder crear entradas y salidas de error en los subVIs.

Los Clusters de Error incluyen un valor booleano que indica si ocurre un error, un código que nos indica el tipo de error que se ha producido, y una cadena de caracteres que indica donde se ha producido el error.

4 Programación Básica

En este apartado, se habla sobre la programación básica que se puede realizar con *Labview* y el modo de realizarla. Las funciones que se mencionan en este apartado, son meramente informativas y no son las únicas que se pueden emplear. Para más información, se recomienda leer la ayuda que proporciona *Labview* sobre cada una de ellas.

4.1 Uso de Bucles

Tanto para la programación en texto como para la programación visual que ofrece *Labview*, es necesario la utilización de bucles que permitan realizar una misma tarea repetitivamente; ya sea mientras se cumpla una condición o durante un tiempo prefijado.

Estos bucles son de gran importancia para el diseño de máquinas de estado y para gestores de colas de mensajes.

Es conocido, que los dos tipos de bucles por excelencia son:

- While: La ejecución se repite mientras se cumpla la condición.

  Para crear un loop while, basta con seleccionarlo en la paleta de funciones y seleccionar la zona que deseamos que se repita. Hay que ajustar la condición y darle a ejecutar.

- For: La ejecución se repite una cantidad fija de veces.

  Se agregan de la misma manera que los loop while. Hay que establecer el número de iteraciones que tiene que realizar el bucle.

En ambos casos, el terminal de iteración, muestra la iteración en la que se encuentra en ese momento y la comunicación entre el interior del bucle y el exterior se realiza a través de túneles.

En la Figura 121 se muestra el ejemplo de un programa con dos bucles diferentes, un bucle For y un bucle While. El bucle For, incrementa el valor del indicador numérico con cada iteración del bucle. El bucle While, realiza una resta en cada iteración mientras el resultado de la misma sea positivo.

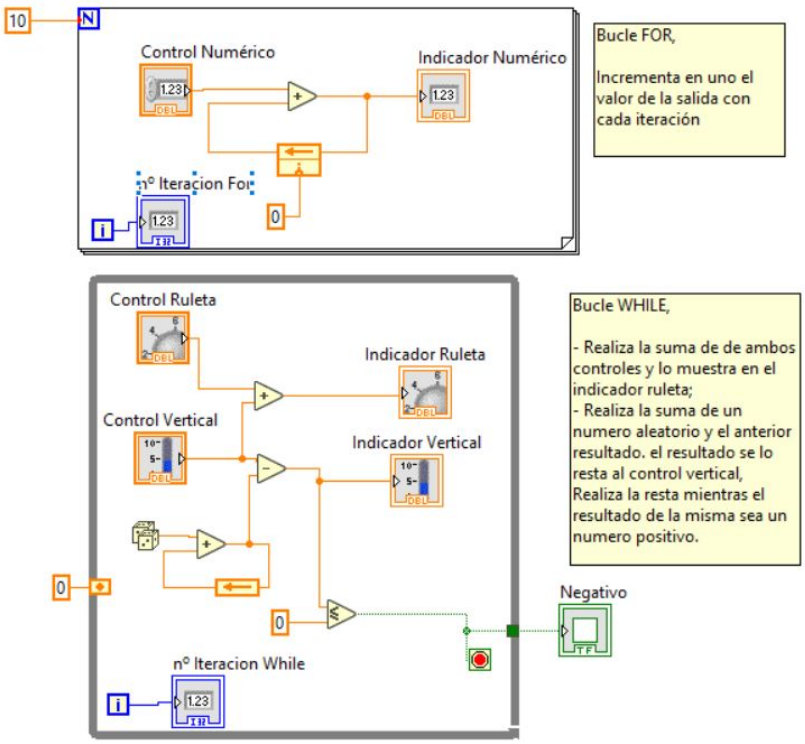


Figura 121: Ejemplo de bucles For y While

Las diferencias fundamentales a la hora de utilizar un bucle u otro son:

Bucle While	**Bucle For**
- Dan el valor de la última iteración. - Detiene la ejecución cuando se cumple la condición. - Debe ejecutarse al menos una vez.	- Dan un array de datos, con los valores de cada iteración. - Se detiene la ejecución cuando se completa el número de iteraciones fijadas. - Puede no tener que ejecutarse.

A ambos tipos de bucles, se les puede añadir temporización del tipo:

- *Wait until Next:* Cuenta una cantidad de ms desde que se inicia el programa y está sincronizada con el reloj del sistema.
- *Wait (ms):* Cuenta la cantidad exacta de ms.
- *VI Express Time Delay:* Simula a wait, pero con Clusters de Error.

- *Elapsed Time:* Interrumpe la ejecución cuando se alcanza el tiempo.

Para pasar datos entre iteraciones de bucle, se emplean los registros de desplazamiento.

Hay cuatro tipos de registros de desplazamiento:

1) Registro No Inicializado.

   Para la primera iteración toma el valor 0, y para las sucesivas iteraciones toma el valor de la anterior iteración. Al volver a ejecutar el programa, el valor de la primera iteración será el valor de la última iteración realizada.

   En la Figura 122 se ve un ejemplo de este tipo de registros.

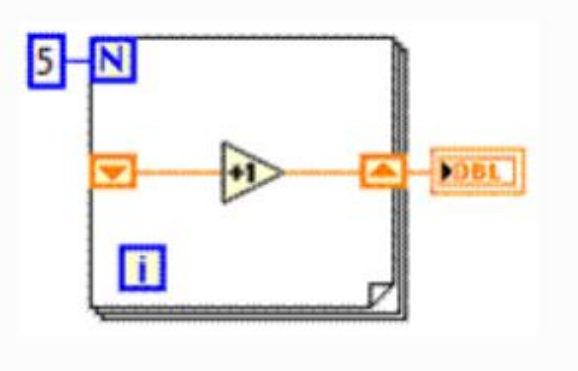


Figura 122: Registro de Desplazamiento No Inicializado

2) Registro Inicializado.

   Al comenzar la ejecución, el registro se inicializa con el valor asignado y para sucesivas iteraciones toma el valor de la anterior iteración. Al volver a ejecutar el programa, el valor de la primera iteración es el valor asignado.

   En la Figura 123, se ve un ejemplo de este tipo de registros.

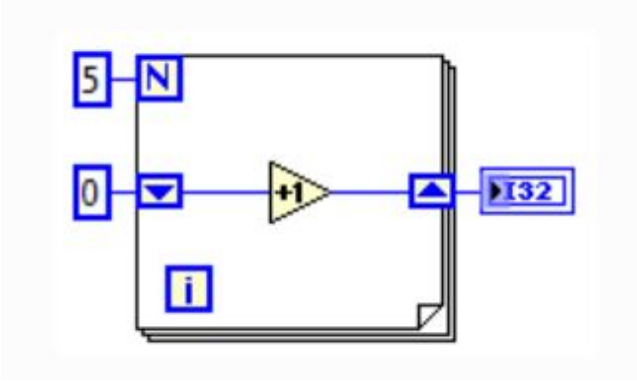


Figura 123: Registro de Desplazamiento Inicializado

3) Múltiples Registros de Desplazamiento.

Cuando tenemos varias entradas y varias salidas, y van emparejadas, se emplean estos registros. Funcionan igual que los registros inicializados.

En la Figura 124, se ve un ejemplo de este tipo de registros.

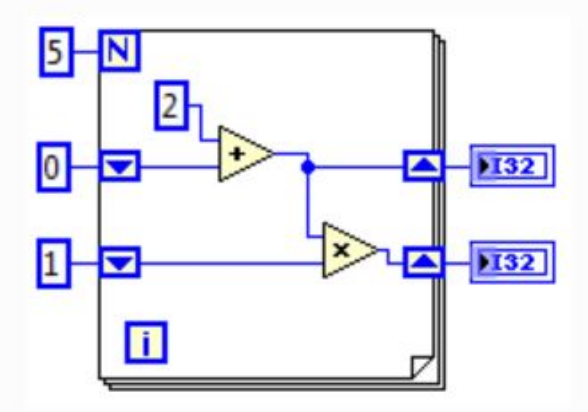


Figura 124: Múltiples Registros de Desplazamiento

4) Registros de desplazamiento apilados.

Con la primera iteración, se usan los valores inicializados; en la segunda iteración se pierde la inicialización inferior y se una el valor de la primera iteración; en la tercera iteración, se usan los valores de la primera y la segunda iteración, y así sucesivamente.

En la Figura 125, se ve un ejemplo de este tipo de registros.

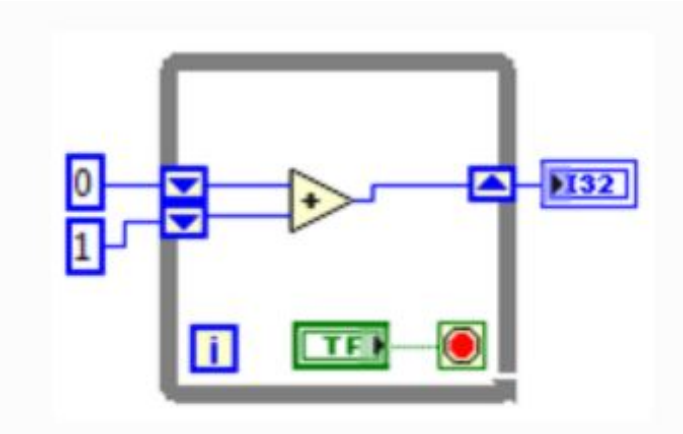


Figura 125: Registros de Desplazamiento Apilados

En [54], explican detalladamente el comportamiento de este tipo de Registros.

4.2 Arrays y Clusters

Las agrupaciones de datos que están relacionados entre sí, dan lugar a los llamados Arrays y Clusters.

I. Arrays.

Se emplean para almacenar datos similares o almacenar valores y poder identificarlos por un índice. El primer índice del array es 0.

Los tipos de datos que podemos almacenar son: booleanos, caracteres y/o numéricos.

Los arrays se pueden crear hasta con 3 dimensiones: fila, columna y profundidad, tal y como se muestra en laFigura 126.

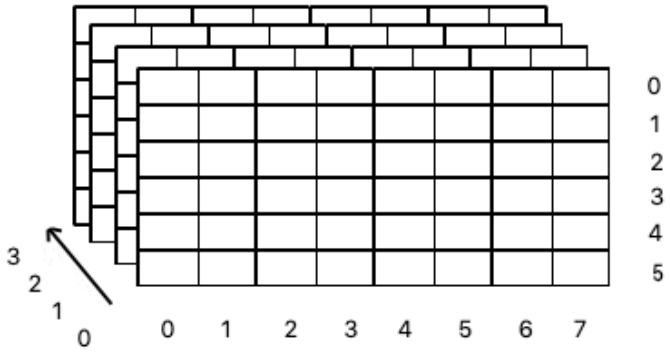


Figura 126: Estructura Arrays

No se puede crear un array de array como sucede en la programación en texto, pero se puede usar un array multidimensional.

Las funciones más comunes para trabajar con array son:

- *Array Size:* Indica el número de elementos del array que están siendo usados. Si el array es multidimensional, indica el número de elementos usados en cada dimensión.
- *Array Subset:* Devuelve una poción de un array. Se solicita el comienzo de la porción y la longitud de la misma.
- *Build Array:* Concatena elementos para formar un array. También concatena varios arrays para hacer otro mayor. Se puede hacer un array más largo o un array 2D con dos arrays 1D.
- *Index Array:* Extrae un elemento o subarray de un array. Se solicita el índice del elemento en cuestión.

Como explican en [55], se debe tener en cuenta que las funciones aritméticas son polimórficas, esto quiere decir que podemos sumar escalares y arrays, siendo el resultado el siguiente:

- Suma dos escalares, resultado otro escalar.
- Suma un escalar y un array, resultado un array. Se suma el escalar a cada elemento del array.
- Suma dos arrays del mismo tamaño, resultado otro array suma elemento a elemento.
- Suma dos arrays de diferente tamaño, resultado otro array del tamaño del pequeño, sumando elemento a elemento.

En la Figura 127 se muestra el polimorfismo de las funciones aritméticas.

En [51] y en [55], se explica detalladamente como se crean arrays de 2D y cuál es la manera adecuada de mostrar los datos de un array con bucles For y con bucles While.

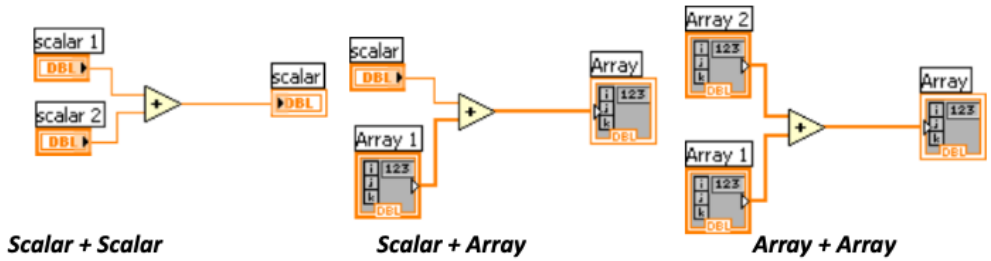


Figura 127: Polimorfismo en funciones aritméticas

En los bucles While, se emplean un indicador u otro dependiendo la manera deseada de mostrar los datos:

- *Indicador Wafeform Graph*, para mostrar todos los datos a la vez.
- *Indicador Waveform Chart* para mostrar los datos de manera continua.

II. Clusters

Son similares a los arrays, solo que los elementos del mismo no tienen que ser del mismo tipo.

- Para visualizar en el DB los Cluster se usa "*UnBundle by Name*" y se selecciona el elemento del Cluster que se desea utilizar.
- Para modificar los datos del Cluster, se usa "*Bundle by Name*" de salida y se selecciona el elemento a modificar.

Cuando se crea un Cluster, es importante hacer definiciones Type o Strict Type, para que se pueda propagar la modificación del Cluster a todos los lugares donde se utiliza el Cluster.

En [55], explican detalladamente el uso de las funciones *Bundle by Name* y *UnBundle by Name.*

En [51] y [55], hay información ampliada sobre lo comentado en este apartado.

4.3 Creación de estructuras y su utilización

Hay dos tipos de estructuras que se emplean en *Labview*, las estructuras Case y la estructura Event. Se emplean principalmente para máquinas de estados.

I. Case

Las partes de la estructura son:

- Etiqueta del Selector: permite navegar por los subdiagramas posibles.
- Terminal del selector de casos: es la entrada de decisión. El tipo de datos, determina el tipo y la cantidad de datos.

Dentro de la estructura Case, se presentan varios tipos:

- Case Booleana: Hay que programar dos opciones: True y False.
- Case Numérica: Es posible trabajar con varios casos diferentes. Para añadir casos se selecciona "*Add case After*". Hay que seleccionar cuál es el caso por defecto seleccionando "*Make this the default case*".
- Case String: Hay que editar los casos y poner los nombres que deseamos.

II. Event

Son estructuras que reaccionan ante eventos asíncronos que provienen del interfaz del usuario, de entradas y/o salidas externas o de otras partes del programa.

Se suelen poner en el interior de bucles While.

5 Programación Básica II

Labview se basa en un principio muy importante a la hora de programar, el concepto de modularidad. Este concepto permite evitar duplicidad de código y facilita la comprensión del mismo.

El concepto de modularidad consiste en dividir el código en partes más simples que tienen funcionalidad por si mismas; esto permite poder implementar estas partes de código en otros proyectos que requieran estas funcionalidades.

Estas partes de código divisibles son los SubVIs.

5.1 SubVI y panel de conectores

Para crear un SubVI, se selecciona la parte del programa que se desea y en el menú *Edit*, se selecciona *Create SubVI*.

En la Figura 128 se puede ver la ventana *Context Help* que muestra las entradas y salidas del SubVI. Esta ventana se obtiene pulsando sobre la interrogación. También se puede observar el panel de conectores en la parte superior derecha, y se puede observar que este SubVI tiene una entrada de tipo DBL (naranja) y dos salidas, una entrada tipo DBL y otra entrada tipo long (azul).

En la parte superior del Panel de Conectores se encuentran los Cluster y en la parte inferior los terminales de los errores si los hubiera.

En la Figura 129 se muestra el icono del SubVI ocupando el espacio de la porción de programa que contiene el SubVI. Este icono sería recomendable que haga referencia al contenido del SubVI, para ello, se puede crear el icono o modificarlo haciendo doble clic sobre el mismo. Se puede componer de texto, símbolos o dibujos, o una combinación de todas o alguna de ellas.

También es interesante en este punto, agregar una descripción de lo que hace el SubVI para poder verlo en la ventana Context Help.

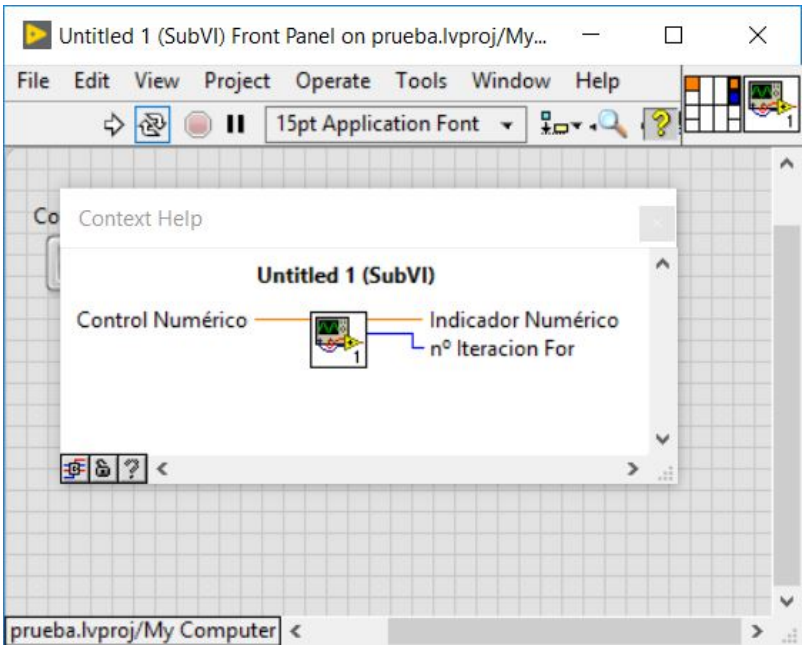


Figura 128: Context Help

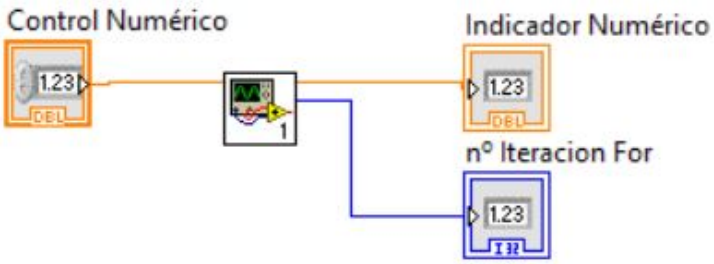


Figura 129: Icono

5.2 Adquisición de medidas

Con *Labview* se puede adquirir todo tipo de datos, desde diferentes tipos de hardware, tanto suministrado por NI como por otros fabricantes.

Labview nos presenta tres opciones de adquisición de datos para hardware de NI:

- Adquisición a través de MAX:

  Permite configurar hardware y software de NI, pudiendo crear y editar canales, tareas, interfaces, escalas e instrumentos virtuales. También permite ejecutar diagnósticos del sistema y paneles de prueba.

  Se puede probar rápidamente la conexión con el DAQ y configurar el Hardware si es necesario.

  En [56], explican detalladamente como instalar un dispositivo hardware empleando MAX y como configurar paneles de prueba.

- Adquisición a través de DAQmx Assistant

  Permite controlar el hardware de manera más precisa y generar un rendimiento óptimo en las medidas. Necesita licencia adicional para trabajar con ello.

  Las partes en las que se realiza la adquisición y los VI asociados a cada una de las partes son:

  [1] Crear la tarea: DAQmx Create Virtual Channel.vi

  [2] Configurar la tarea: DAQmx timing (sample clock).vi y DAQmx Reference Trigger (Digital Edge).vi

  [3] Iniciar la tarea: DAQmx Start Task.vi

  [4] Adquirir o generar datos: DAQmx read.vi o DAQmx Write.vi

  [5] Borrar la tarea: DAQmx clear.vi

  [6] Verifica Errores: Simple error hadler.vi

La mejor manera de programar con *Labview* y no cometer errores difíciles de encontrar es seguir al realizar la adquisición con DAQmx es seguir los pasos descritos anteriormente.

- Adquisición a través de un VI

  Es la opción en la que se programa íntegramente todo el sistema de adquisición.

Si el hardware con el que se pretende realizar la adquisición no es de NI, hay que realizar la conexión a través de VISA.

Lo primero que se debe hacer es comprobar que se tiene conexión con el dispositivo e instalar el controlador necesario para la comunicación.

Los pasos a seguir en la adquisición empleando VISA son los siguientes:

[1] Abrir sesión para instrumentos: visa open.vi

[2] Realizar operaciones de entrada/salida: visa read.vi o visa write.vi

[3] Cerrar sesión para instrumentos: visa close.vi

[4] Verificar errores: simple error handler.vi

Para mayor información de los VI mencionados en este apartado, visualizar la ayuda de cada uno de ellos dentro del programa.

5.3 Archivos: creación y uso

En *Labview* se puede trabajar con tres tipos de archivos: ASCII, TDMS, y Binario.

Se podría realizar una clasificación de utilización del tipo de archivos según algunos criterios:

- Para precisión numérica: Es mejor emplear archivos Binarios y TDMS que ASCII.
- Para compartición de datos: El más óptimo con cualquier programa es ASCII, el TDMS es mejor con programas de NI y Excel, y el Binario proporciona información detallada del formato.
- Según la eficiencia: TDMS y Binario son las más optimas y ASCII es buena.

En conclusión, el uso ideal de cada tipo de archivos es el siguiente:

- ASCII: Compartir datos con otros programas cuando el tamaño y la precisión numérica no son importantes.
- TDMS: Almacenar datos de mediciones y metadatos relacionados. Es capaz de transmitir datos a altas velocidades sin perdidas de precisión.

- Binario: Almacena datos numéricos de forma compacta con capacidad de acceso aleatorio.

En la Figura 130 se muestra la secuencia típica a la hora de trabajar con archivos, sean del tipo que sean. Primero se debe abrir o crear el archivo, para posteriormente poder trabajar sobre él. Cuando se termina de trabajar, se debe cerrar el mismo y verificar si han ocurrido errores.

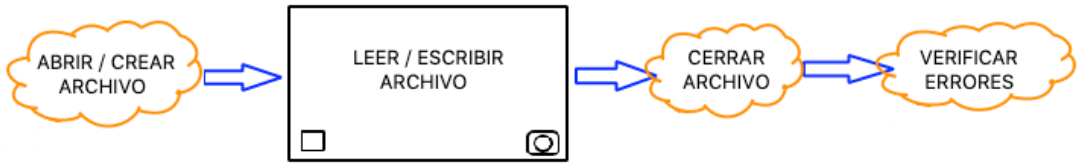


Figura 130: Secuencia de utilización de un archivo

Existen funciones en la Paleta de funciones en el menú "*Programming" / "File I/O"* que permiten poder realizar todas las operaciones comentadas. Se pueden distinguir dos clases de funciones:

- Fondo blanco: funciones de E/S de alto nivel. Se emplea para archivos que tienen formato.
- Fondo amarillo: funciones de E/S de bajo nivel. Se emplean cuando el archivo tiene texto plano o binario.

En ambos casos, cuando se crea o se abre un archivo, se asocia un Refnum para que el resto del programa sepa a qué archivo hace referencia. Cuando se cierra el archivo, el Refnum desaparece.

Algo importante a la hora de trabajar con archivos en *Labview* es crear rutas de archivos y carpetas. Hay dos clasificaciones de rutas:

- Absoluta: Describe la ubicación desde el nivel superior del sistema de archivos.
- Relativa: Describe la ubicación en relación con una ubicación arbitraria en el sistema de archivos.

Para seguir la filosofía de modularidad de *Labview*, es recomendable emplear la clasificación relativa y así no tener que cambiar la ubicación cada vez que se emplea el VI.

Las funciones "Application Directory VI" y "Get System Directory VI" son para crear rutas relativas. La primera devuelve la ruta del directorio donde se encuentra la aplicación; y la segunda devuelve el directorio de sistema especificado a la entrada del tipo de directorio de sistema, por lo que hay que introducir la carpeta donde se quiere guardar.

Para la creación de carpetas, existe la función "*create Folder*", pero es interesante verificar si dicha carpeta existe con las funciones "*VI Check if File*" o "*folder Exists*" para que no ocurran errores posteriormente.

Si se quieres guardar varios archivos con nombres relacionados dentro de una carpeta, por ejemplo, para almacenar datos de un sensor en diferentes días, se emplean nombres de archivos dinámicos, y para ello, se emplea el uso de bucles FOR.

A los archivos se les puede añadir un encabezado, para ello se emplean distintas técnicas:

1) *Hardcoding*: se introducen encabezados con tabulaciones y finales de línea.
2) Uso de subVIs: se procesa el formato. No se tiene que modificar el código.
3) Escritura en varios canales: se crea un array bidimensional en una tabla de cadenas de caracteres.
4) Lectura de datos del canal: Lee solo los datos y se salta los encabezados

Para poder profundizar en las funciones y técnicas de programación emplear la ayuda de *Labview*.

5.4 Programación de equipos secuenciales y de estado

Como explican en [57], existen numerosos modelos de diseño para VIs de *Labview* y la mayoría de aplicación emplean al menos uno de ellos. En este apartado se explora el modelo de diseño de la máquina de estados.

El objetivo de este apartado es usar el flujo de datos para asegurar la ejecución secuencial de nodos y no basarse en una estructura concreta.

- Programación Secuencial:

  Hay que tener en cuenta que es lo primero que se debe ejecutar. Para ello se realiza una estructura de bucle y se usan los mensajes de error.

  De este modo, no se ejecuta el bucle hasta que no llega el mensaje de error, dando prioridad al orden de ejecución en la programación y haciéndola secuencial.

  Se puede usar un marco para hacer que la ejecución sea secuencial, ejecutándose de izquierda a derecha con la función Sequence. Estas estructuras son beneficiosas porque garantizan el orden de ejecución, pero prohíben la ejecución en paralelo del programa.

  Se deben usar en los casos en los que la ejecución necesite un orden por la dependencia de los datos entre los nodos y no se pueda parar la aplicación antes de que termine la ejecución.

El principal problema de las estructuras Sequence es que no tienen control de errores, por lo que es más recomendable emplear estructuras Case y controlar el orden de ejecución a través del flujo de datos.

Se emplean estructuras tipo CASE y tipo WHILE.

- Programación de estados:

  Hay que realizar un diagrama de transición de estados en el que aparezca:

  (a) Cambio de orden de la secuencia

  (b) Repetición de elementos de la secuencia

  (c) Ejecución del elemento cuando se cumpla una determinada condición

  (d) Detención del programa inmediato en lugar de esperar a que termine la ejecución.

  Este diagrama es muy similar al que se realiza en programación en C, está compuesto por los estados, las transiciones, un punto de partida y un punto de fin.

- Máquina de estados:

  Están diseñadas para realizar cíclicamente una o varias acciones entre el inicio y el fin de la ejecución.

  La infraestructura de una máquina de estados en *Labview* es la siguiente:

  - Registro de desplazamiento de los datos de estado

    Almacena los datos que usan los diferentes estados de la máquina. Hay que actualizar estos datos en el código de funcionalidad del estado para que los cambios se puedan compartir con otros estados.

  - Cluster de datos de estado con tipo definido

    En este Cluster se definen los datos que se usaran en el código de funcionalidad del estado para cada estado de la máquina.

  - Enum de estados con tipo definido

    Se utiliza para crear una lista de estados posibles de la máquina.

  - Bucle While

Implementa el flujo del diagrama de transición de estados

- Registro de desplazamiento del estado

  Almacena el valor del próximo estado que se debe ejecutar y pasa dichos datos al selector de la estructura CASE durante la siguiente iteración del bucle while.

- Estructura Case

  Cada estado está representado en un caso de la estructura CASE.

En la Figura 131 se muestra la infraestructura básica de una máquina de estados construida con *Labview*.

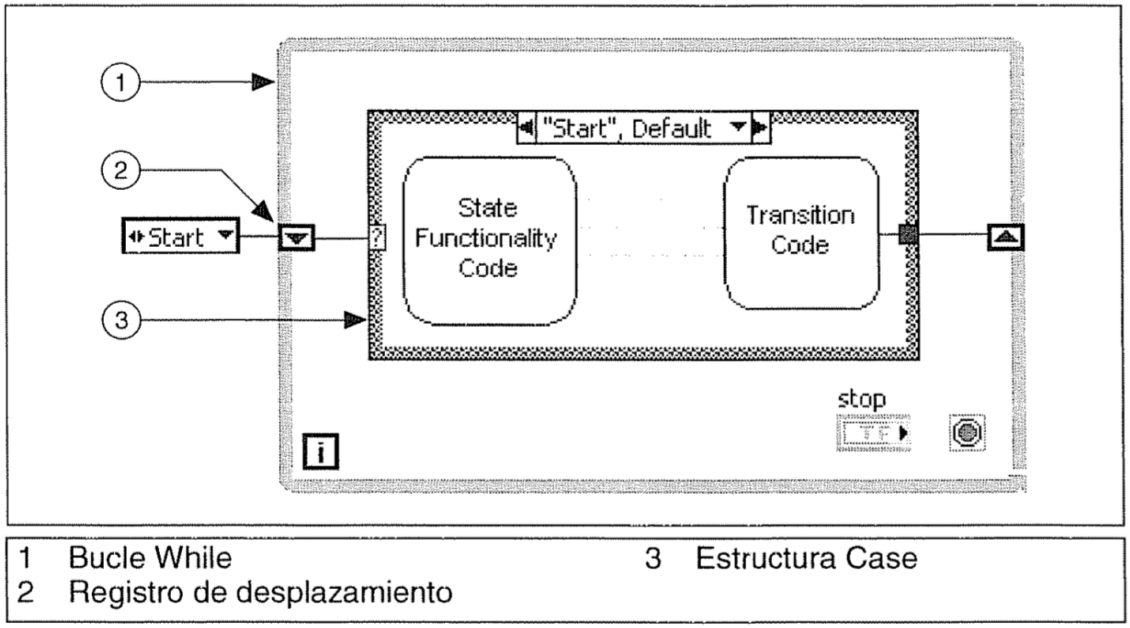


Figura 131: Estructura básica de una máquina de estados en Labview

Las transiciones en la máquina de estados dependerán:

1. Transición predeterminada individual

   Se trata del estado inicial. En él se inicializan las variables fijas, se establece el error y se apunta al siguiente estado.

2. Transición de la función Select

   Se adquieren los datos y de evalúa a que estado hay que ir, dependiendo del valor de error.

3. Transición de la función Case

   Depende del caso en el que se encuentra la máquina, el siguiente estado está en cada caso.

4. Array de transición

   En vez de utilizar la estructura Case, se obtiene el siguiente estado a través de un array.

También se pueden generar máquinas de estado basadas en eventos. En ese caso, hay que esperar a que suceda el evento para saber en qué estado se encuentra la máquina.

- Paralelismo

  Puede existir paralelismo entre tareas que se ejecutan en un mismo VI, para ello, basta con colocar en diferentes bucles lo que se quiere que se ejecute paralelamente y que las señales de salida de un bucle no sean las señales de entrada de otros bucles.

Bibliografía

[1] S. U. K. R. Z. S. K. Rafiullah Khan, «Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges,» de *2012 10th International Conference on Frontiers of Information Technology*, Islamabad, 2012.

[2] M. Martinez, «La piedra del sísifo,» 19 12 2016. [En línea]. Available: http://lapiedradesisifo.com/2016/12/19/iot-tostadora/.

[3] Internet of Things Value Ceation Network, «Internet of Things Value Ceation Network,» [En línea]. Available: http://www.internet-of-things.no/iot.html.

[4] UIT, «Visión general de la Internet de las cosas,» 2012.

[5] Towards a definition of the Internet of Things (IoT), «IEEE Internet of Things,» Mayo 2015. [En línea]. Available: https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Issue1_14MAY15.pdf.

[6] ETSI, «Machine-to-Machine communications (M2M); Functional architecture,» 2013.

[7] ETSI, «oneM2M Requirements,» 2016.

[8] RFID and the Inclusive Model for the Internet of Things, «docbox.etsi.org,» 11 2008. [En línea]. Available: https://docbox.etsi.org/zArchive/TISPAN/Open/IoT/low%20resolution/www.rfidglobal.eu%20CASAGRAS%20IoT%20Final%20Report%20low%20resolution.pdf.

[9] J. Gold, «NetWork World España,» [En línea]. Available: https://www.networkworld.es/archive/que-es-el-iot-industrial-y-por-que-las-apuestas-son-tan-altas.

[10] A. Llanos, «oasys,» [En línea]. Available: https://oasys-sw.com/diferencias-iiot-e-iot/.

[11] G. T. V. M. P. Wu Rong, «The internet of things (IoT) and transformation of the smart factory,» de *2016 International Electronics Symposium (IES)*, Denpasar, 2016.

[12] Estandarizacion para la industria 4.0, «UNE,» [En línea]. Available: https://www.une.org/normalizacion_documentos/Estandarizacion-para-la-industria-4_0.pdf.

[13] Optical Networks, «Optical Networks,» [En línea]. Available: https://www.optical.pe/internet-industrial-de-las-cosas-iiot/.

[14] P. C. Báscones, «Comillas Universidad Pontificia,» [En línea]. Available: https://www.comillas.edu/es/blog-catedra-industria-conectada/tendencias-iiot-y-ciberseguridad?jjj=1551783287457.

[15] O. Damian. [En línea]. Available: http://www.academia.edu/8013013/Cap%C3%ADtulo_5_El_paradigma_cliente-servidor.

[16] J. L. Juarez Manzano, «Vintegris,» [En línea]. Available: http://vintegris.info/protocolos-iot-001/.

[17] I. Porro Saez, «Incibe-cert,» 07 02 2019. [En línea]. Available: https://www.incibe-cert.es/blog/iot-protocolos-comunicacion-ataques-y-recomendaciones.

[18] Restfulapi, «REST API Tutorial,» [En línea]. Available: https://restfulapi.net/.

[19] Github, «CoAP,» [En línea]. Available: http://coap.technology.

[20] Github, «XMPP,» [En línea]. Available: https://xmpp.org.

[21] Object Management Group, Inc, «DDS,» [En línea]. Available: https://www.omgwiki.org/dds/.

[22] H. Mishra, «IoTbyHVM,» [En línea]. Available: https://iotbyhvm.ooo/stomp/.

[23] OASIS, «AMQP,» [En línea]. Available: https://www.amqp.org.

[24] Camara Valenciana, «Tecnologia para los negocios,» [En línea]. Available: https://ticnegocios.camaravalencia.com/servicios/tendencias/caminar-con-exito-hacia-la-industria-4-0-capitulo-14-dispositivos-i-internet-de-las-cosas-iot/.

[25] Optical Networks, «Optical Networks,» 2018. [En línea]. Available: https://www.optical.pe/ciberamenaza-en-los-dispositivos-iot/.

[26] D. Delgado, «Arquitectura de Software,» 3 12 2014. [En línea]. Available: http://arquitecturasomos4.blogspot.com/2014/12/publish-subscriber.html.

[27] National Instrument, «NI - LabView,» [En línea]. Available: http://www.ni.com/es-es/shop/labview.html.

[28] NI, «NI Academico / Datos y Estructuras,» [En línea]. Available: http://www.ni.com/academic/students/learnlabview/esa/datatypes.htm.

[29] NI, «National Intrument,» 03 2017. [En línea]. Available: http://zone.ni.com/reference/en-XX/help/371361P-01/lvconcepts/coercion_dots/.

[30] National Instrument , «NI - soporte,» [En línea]. Available: http://www.ni.com/es-es/support/downloads/drivers.html.

[31] NI, «NI Academico / Depuración,» [En línea]. Available: http://www.ni.com/academic/students/learnlabview/esa/debugging.htmhttp://www.ni.com/academic/students/learnlabview/esa/debugging.htm.

[32] NI, «NI Academico / Estructuras en ejecución,» [En línea]. Available: http://www.ni.com/academic/students/learnlabview/esa/execstructures.htm.

[33] E. Jullan- Laime y A. Almidon, «Manual de programacion Labview 9.0,» 10 2018. [En línea]. Available:

https://www.researchgate.net/publication/328404013_Manual_de_programacion_LabVIEW_90.

[34] NI, «Instalar Controladores de Hardware,» [En línea]. Available: http://www.ni.com/academic/students/learnlabview/esa/hardware.htm.

[35] Univeridad del pais vasco, «Open Course Ware,» [En línea]. Available: https://ocw.ehu.eus/file.php/54/MATERIALES_DE_ESTUDIO/T8/T8.pdf.

[36] HIveMQ, «HiveMQ,» Enero 2015. [En línea]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/.

[37] ARROW, «Arrow,» [En línea]. Available: https://www.arrow.com/es-mx/research-and-events/articles/protocols-for-the-internet-of-things.

[38] HIVEMQ, «HIVEMQ MQTT Essentials,» Julio 2019. [En línea]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/.

[39] HIVEMQ, «MQTT ESSENTIALS,» febrero 2015. [En línea]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/.

[40] HiveMQ, «HiveMQ,» Febrero 2015. [En línea]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/.

[41] HIVEMQ, «MQTT ESSENTIALS,» febrero 2015. [En línea]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/.

[42] ISO, [En línea]. Available: https://www.iso.org/standard/69466.html.

[43] OASIS, «MQTT Version 3.1.1,» abril 2014. [En línea]. Available: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.html.

[44] OASIS, «MQTT VEraion 5.0,» [En línea]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[45] J. Levell, «MQTT.org,» [En línea]. Available: http://mqtt.org/2019/04/mqtt-v5-0-now-an-official-oasis-standard.

[46] M. Tolia, «Moulin Tolia,» Enero 2019. [En línea]. Available: https://mntolia.com/10-free-public-private-mqtt-brokers-for-testing-prototyping/.

[47] eclipse paho, «paho,» [En línea]. Available: https://www.eclipse.org/paho/.

[48] eclipse fundation, «eclipse paho,» [En línea]. Available: https://projects.eclipse.org/projects/iot.paho/downloads.

[49] R. Light, «libmosquitto man page,» [En línea]. Available: https://mosquitto.org/man/libmosquitto-3.html.

[50] Indie-Energy, «Git Hub,» 2017. [En línea]. Available: https://github.com/Indie-Energy/AWS-IoT-RESTful.

[51] Pete1982, «GitHub,» Enero 2017. [En línea]. Available: https://github.com/DAQIO/LVMQTT.

[52] HiveMQ, «HiveMQ,» Julio 2019. [En línea]. Available: https://www.hivemq.com/blog/seven-best-mqtt-client-tools/.

[53] MQTT.fx, «MQTT fx,» 2019. [En línea]. Available: https://mqttfx.jensd.de.

[54] J.-P. Mens, «Jan-Piet Mens,» 2011. [En línea]. Available: https://jpmens.net/2013/11/19/mqtt-inspector-for-ios/.

[55] Google Play, «MyMQTT,» [En línea]. Available: https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client&hl=en_US.

[56] workswithweb, «MQTT Toolbox,» Agosto 2016. [En línea]. Available: https://www.hivemq.com/blog/mqtt-toolbox-mqttbox/.

[57] ECLIPSE MOSQUITTO, [En línea]. Available: https://mosquitto.org/download/.

[58] Eclipse, «Eclipse Mosquitto,» [En línea]. Available: https://mosquitto.org.

[59] M. Rouse, «TeachTarget,» [En línea]. Available: https://searchdatacenter.techtarget.com/es/definicion/Internet-Industrial-de-las-Cosas-IIoT.

[60] W. Chirinos, «Paraquesirve,» [En línea]. Available: https://paraquesirve.tv/servidor/.

[61] Inductive Automation, «Inductive Automation.,» [En línea]. Available: https://inductiveautomation.com/.

[62] Ignition IIoT Arquitecture, «Ignition Automation,» [En línea]. Available: https://inductiveautomation.com/static/pdf/IgnitionArchitecture-IIoT-CloudRedundant.pdf.

[63] eclipse paho, «phyton client - documentation,» [En línea]. Available: https://www.eclipse.org/paho/clients/python/docs/.

Tabla de Figuras

Universidad
de Alcalá